VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

RDF ÚLOŽIŠTĚ V PHP

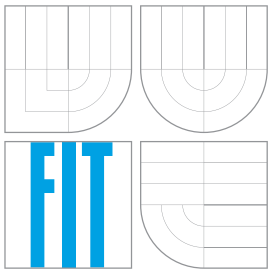
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ RAŠKA

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

RDF ÚLOŽIŠTĚ V PHP

RDF REPOSITORY IN PHP

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ RAŠKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2011

Abstrakt

Tato práce se zabývá návrhem a implementací RDF úložiště v jazyku PHP. Řeší ukládání datového modelu RDF v relační databázi a implementaci podmnožiny dotazovacího jazyka SPARQL. Součástí jsou testy ověřující funkčnost implementovaného RDF úložiště a testy výkonnosti.

Abstract

This Bachelor thesis describes a problem of design and implementation a RDF repository in PHP. This thesis solves storing data model RDF in relational database and solves implementation of subset query language SPARQL. Part of this thesis are performance and functional tests with implemented RDF repository.

Klíčová slova

RDF úložiště, SPARQL, Sémantický Web, PHP, MySQL.

Keywords

RDF repository, SPARQL, Semantic Web, PHP, MySQL.

Citace

Jiří Raška: RDF úložiště v PHP, bakalářská práce, Brno, FIT VUT v Brně, 2011

RDF úložiště v PHP

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny a publikace, z nichž jsem čerpal.

.....
Jiří Raška
9. května 2011

Poděkování

Rád bych poděkoval mému vedoucímu Ing. Radku Burgetovi, Ph.D. za ochotu při konzultacích. Dále bych chtěl poděkovat mým blízkým za pomoc a podporu.

Obsah

Kapitola 1

Úvod

Když Tim Berners-Lee[1] v roce 1989 navrhoval službu WWW, sám netušil, že se jeho projekt v následujících letech stane zaměstnáním, zábavou či každodenní činností pro více než čtvrtinu populace[2]. Se vzrůstajícím počtem uživatelů a nástupem nových technologií došlo a stálé dochází k výraznému přibývání obsahu na WWW a to nejen textových dokumentů, ale i multimediálního obsahu. Dnešní web má také proto podobu nekontrolovatelně rostoucí haldy dokumentů různé kvality a věrohodnosti.

Informace resp. data v těchto dokumentech jsou většinou srozumitelná lidem avšak ne strojům. Vzhledem k nepřebernému množství WWW obsahu je manuální zpracování stále více náročnější až nemožné.

Odpověď na tento problém nabízí článek The Semantic Web[3] v Scientific American z května roku 2001. V tomto článku Tim Berners-Lee, Jim Hendler a Ora Lassila nastínili ideu Sémantického Webu, kde softwaroví agenti budou na webu zpracovávat informace a budou za lidi vykonávat různé úkony. Jako objednání k lékaři na základě bydliště pacienta s ohledem na hodnocení lékaře, dopravní spojení atp.

Základem Sémantického Webu je reprezentace sémantiky dat v jednotném formátu, které bude možné strojově zpracovat. Tímto modelem dat je technologie RDF, založená na popisu objektů pomocí RDF trojice(subjekt-predikát-objekt). Tato práce se zabývá uložením RDF trojice v relační databázi a implementací podmnožiny dotazovacího jazyka SPARQL nad tímto RDF úložištěm v jazyku PHP.

Text této práce je strukturován do 6 kapitol, první tři kapitoly se zabývají technologiemi Sémantického Webu, zejména RDF a SPARQL a zbylé tři návrhem, implementací a testováním RDF úložiště.

Součástí tohoto dokumentu je také příloha, ve které se nachází příklady použití knihovny RDF úložiště, veškeré provedené testy ověřující funkčnost a kompletní testy výkonnosti, jenž byly provedeny nad RDF úložištěm.

[1] http://www.w3.org/People/Berners-Lee/
[2] Statistické údaje populace užívající internet http://www.internetworldstats.com/stats.htm
[3] http://www.scientificamerican.com/article.cfm?id=the-semantic-web

Kapitola 2

Sémantický Web

Tato kapitola seznamuje čtenáře se Sémantickým Webem a jeho technologiemi.

2.1 Co je Sémantický Web?

Ideu Sémantického webu poprvé prezentoval[2] v roce 2001 zakladatel WWW Tim-Berners-Lee se svými spolupracovníky. Sémantický Web prezentoval na příkadu, kde každý člověk bude mít softwarového agenta. Takovýto softwarový agent potom bude schopný domluvit schůzku s lékařem na základě diagnózy, časového harmonogramu, bydliště uživatele případně dalších preferencí jako cena či hodnocení lékaře.

Na současném webu je tato idea nerealizovatelná, avšak s nasazením technologií Sémantického Webu bude takovýto scénář možný. Uplynulo již deset let od vyřčení této idee a stále jsme jen na počátku cesty. Je standardizován datový model, existují základní slovníky, vzniklo několik dotazovacích jazyků a jejich implementací. Ta nejdůležitější část však stále chybí a to publikace sémantických dat na WWW. Bez ní je nemožná poslední část, která je však našim cílem.

2.2 Technologie Sémantického Webu

Sémantický Web[1] je možné označit jako rozšíření současného webu o metadata. Metadata jsou strukturovaná data o datech. Příkladem je řetězec Petr, tento řetězec může být interpretován odlišně různými uživateli a to především na základě jejich vzdělání, natož stroji. Pokud však dodáme tomuto řetězci metadata, jenž definují význam a znakovou sadu, potom každý stroj může takovýto řetězec korektně zpracovat, sdílet jej a znovupoužívat.

Zpracovávání metadat má být automatizováno prostřednictvím softwarových *agentů*, což by měli být autonomní programy, které obdrží úkoly od svých uživatelů a budou procházet WWW, komunikovat s dalšími *agenty*, porovnávat informace na základě uživatelských preferencí a následně předloží uživateli odpověď. Úkolem *agenta* ovšem není nahrazení člověka na Sémantickém Webu, ale pouze vykonaní určitých rutin a nabídnout uživateli možnosti dle jeho preferencí. Konečné rozhodnutí tak zůstane na člověku.

Univerzálním jazykem pro reprezentaci metadat je XML. Tento značkovací jazyk svými vlastnostmi nejvíce odpovídá požadavkům na otevřenost, přenositelnost a interoperabilitu dat, avšak má určitá omezení týkající se sémantiky vnořených elementů.

V prostředí WWW potřebujeme popisovat objekty a vztahy mezi nimi. K tomuto popisu byl vytvořen a standardizován organizací W3C datový model RDF. RDF poskytuje

jednoduchý a zároveň silný nástroj pro popis zdrojů. Zdrojem se zde rozumí cokoliv, co je dosažitelné prostředníctvím sítě WWW. RDF je základním stavebním kamenem Sémantického Webu a také základem této práce, proto je mu vyhrazena samotná kapitola.

Mezi objekty, které popisujeme prostřednictvím RDF, jsou různé vztahy jako název, věk, datum atp. Tyto vztahy jsou znalostmi, které jsou shromažďovány do oborových domén *ontologiích*. Pojem *ontologie* původně pochází z filosofie, z pohledu výpočetní techniky jde o popis určité problematiky prostřednictvím formální a deklarativní reprezentace. *Ontologie* je slovník znalostí určité domény. V oblasti již vytvořených *ontologiích* je na tom Sémantický Web poměrně dobře, v této práci některé ontologie používám, tak je proto zde zmíním. Pro popis osob, jejich vlastností či vztahů k jiným osobám byla vytvořena ontologie *FOAF*(The Friend Of A Friend). Nejčastěji používanou ontologií, která definuje základní vlastnosti zdrojů je *Dublin Core* s vlastnostmi např. *title*, *creator*, *subject* nebo *date*.

Úlohou Sémantického Webu je v podstatě přenesení strukturovanych dat, která jsou nejčastěji uložena v databázi, do popředí v jednotném formátu. Potom lze nad těmito Sémantickými daty provádět dotazy, jak nad daty v databázi. Pro datový model RDF proto vznikly také různé dotazovací jazyky. Vedle jazyka SPARQL, kterému je věnována samostatná kapitola, to jsou např. SeRQL, ARQ, Versa, XsRQL, TRIPLE a Xcerpt. Hlavním cílem těchto dotazovacích jazyků je získávání RDF dat z RDF úložiště.

Kapitola 3

RDF

3.1 Úvod do RDF

RDF(Resource Description Framework)[3] byl vypracován organizací W3C a je základním stavebním prvkem Sémantického Webu. Ačkoliv je někde označován jako jazyk, jde o datový model pro popis, výměnu a znovupoužití metadat na WWW. Záleží tak na aplikaci, jak takový datový model bude ukládat. Pro textové vyjádření je nutné datový model serializovat a RDF má několik serializací např. do XML nebo N3. RDF klade důraz na jednoduchost vyjadřování a jednoduché automatické zpracování stroji.

RDF je založen na myšlence popisu zdrojů. Zdroje jsou v prostředí WWW jednoznačně identifikovány prostřednictvím *URIs*(Uniform Resource Identifiers) a jsou popisovány pomocí tvrzení, ve kterých vedle zdrojů vystupují také již definované znalosti. Více tvrzení je následně propojeno a tvoří tak graf komplexně popisující jeden zdroj nebo třeba celý web.

Prostřednictvím RDF lze tak snadno popisovat jakékoliv zdroje v rámci WWW a tento popis vyjádřit strojově čitelným způsobem. Toho lze potom využít v různých aplikačních oblastech např. ve vyhledávání, katalogizaci, zjištění vztahů mezi zdroji, sdílení a výměně znalostí, hodnocení obsahu nebo pro popis intelektuálních vlastnických práv.

3.2 Tvrzení v RDF

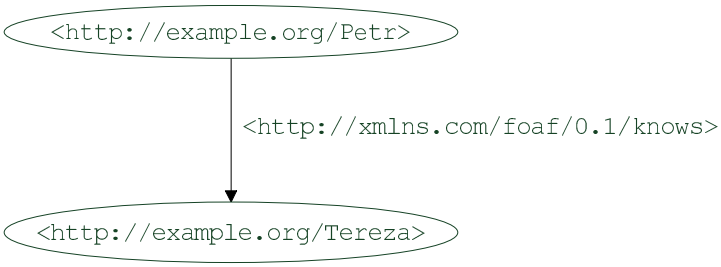
Tvrzení je v RDF tvořeno *subjektem*(subject), *predikátem*(predicate) a *objektem*(object). *Subjekt* je nějaký zdroj o kterém je proneseno tvrzení, *predikát* je vlastnost zdroje většinou již definovaná znalost a *objekt* je hodnotou této vlastnosti. Tvrzení je tvořeno třemi částmi, proto je také běžně užíváno pojmenování *trojice*.

Jako příklad uvedu výrok: „Petr zná Terezu“. V tomto výroku máme subjekt Petr, vlastnost znát a hodnota této vlastnosti je Tereza.

Subjekt, predikát a objekt jsou v RDF reprezentovány pomocí *URI*. Objekt může navíc být i literál, v takovém případě však není možné jej použít jako subjekt v jiném výroku. V tomto příkladu je celá trojice reprezentována pomocí *URI* a tvrzení v Turtle syntax s využitím URI je uvedeno v př. 3.1.

```
<http://example.org/Petr> <http://xmlns.com/foaf/0.1/knows> <http://example.org/
Tereza> .
```

Listing 3.1: RDF tvrzení v Turtle.



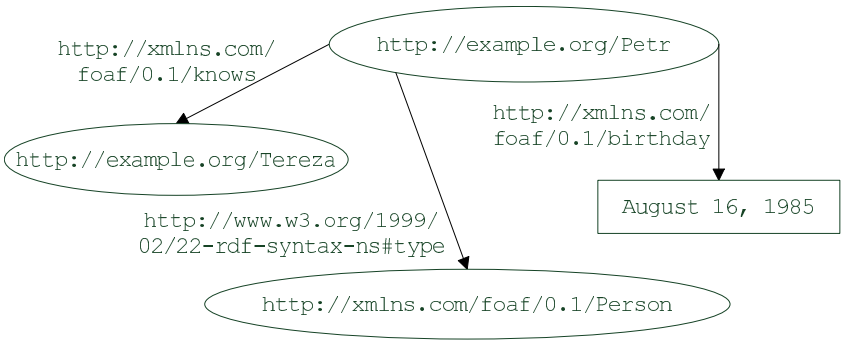
Obrázek 3.1: Jednoduché tvrzení v RDF.

Tvrzení lze vyjádřit i vizuální podobou a to pomocí grafu, kde subjekt a objekt jsou uzly a predikát hranou grafu. Vzorové tvrzení v podobě grafu je vidět na obr. 3.1.

Výše byl popsán subjekt pomocí jednoho výroku, těchto výroků však může být více a tím může být popis komplexnější. Více takovýchto výroků tvoří rosáhlejší graf, kde subjekty a objekty jsou uzly a predikáty jsou hrany grafu. Dále platí, že URI mají podobu oválu a literály obdelníku. Proto mějme tři výroky o jednom subjektu: „Petr zná Terezu“, „Petr má narozeniny 16. října 1985“ a „Petr je osoba“. Tyto výroky lze zapsat jak textově v Turtle syntax př. 3.2, tak i v grafové reprezentaci obr. 3.2.

```
<http://example.org/Petr>  <http://xmlns.com/foaf/0.1/knows> <http://example.org/
Tereza> .
<http://example.org/Petr> <http://xmlns.com/foaf/0.1/birthday> ‘‘August 16, 1985’’ .
<http://example.org/Petr> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type< <http://
xmlns.com/foaf/0.1/Person> .
```

Listing 3.2: Tři RDF výroky v Turtle.



Obrázek 3.2: Tři RDF výroky v grafu.

3.3 Zkrácený zápis URI pomocí prefixů

Zdroje v RDF tvrzeních jsou zapisovány prostředncitvím URI, které však pro zápis není ideální a snadno tak vznikají příliš dlouhé řádky, ve kterých je opakován neustále stejný podřetězec. Proto existuje zkrácený zápis, který je používán v tomto dokumentu dále. Jak v jazyku XML existují prefixy a jmenné prostory, tak i v RDF lze definovat jméno prefixu a jmenný prostor URI.

Z konvence jsou některé prefixy využívány pro konkrétní URI jde především o všeobecně známé a používané ontologie. Např. pro ontologii *Dublin Core* je užíván prefix *dc*, pro ontologii *The Friend Of A Friend* je ustáleným prefixem *foaf*.

Prefix je vždy na začátku dokumentu nadefinován svým názvem, který slouží jako jednoznačný identifikátor a dvojtečkou je oddělen od jmeného prostoru URI. V trojici se potom použije název prefixu oddělený dojtečkou od místního jména v rámci daného jmenného prostoru. Při užití prefixů podoba v Turtle syntax uvedena v př. 3.3.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.org/> .

ex:Petr foaf:knows ex:Tereza .
ex:Petr foaf:birthday "August 16, 1985" .
ex:Petr rdf:type foaf:Person .
```

Listing 3.3: Tři RDF tvrzení s prefixem v Turtle.

3.4 Anonymní uzly

Objektem výroku nemusí být URI nebo literál, ale speciální *anonymní uzel*(Blank node). Takovýto uzel nenese žádnou hodnotu a slouží k strukturalizaci určitých dat. Ovšem je potřeba jej jednoznačně identifikovat, neboť na něj může a je odkazováno v rámci grafu.

Smyslem tohoto konceptu je strukturovat určitá data do více samostatných uzlů. Pokračuji v příkladu výše a den narození Petra, který je nyní vyjádřen pomocí jednoho uzlu, rozdělím do 3 uzlů. To znamená, že na místě stávajícího literálu s datem bude prázdný uzel, který bude sloužit jako subjekt v trojicích tvdících den, měsíc a rok narození.

K dosažení takového cíle použiji RDF anonymní neboli prázdné uzly. Popsané použití lze vidět na př. 3.4 v textové podobě a na obr. 3.3 v grafové ilustraci.

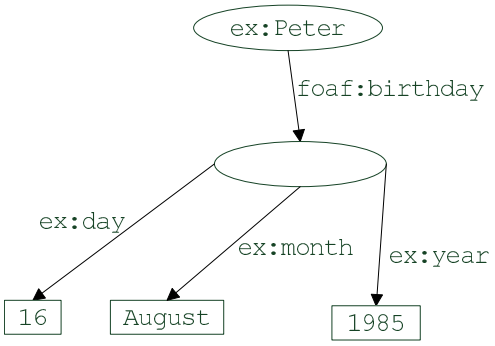
```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.org/> .

ex:Petr foaf:birthday _:petr_bd .
_:petr_bd  ex:day "16" .
_:petr_bd ex:month "August" .
_:petr_bd ex:year "1985" .
```

Listing 3.4: RDF tvrzení s anonymním uzlem v Turtle.

3.5 Typy literálů

Objekt v RDF může být *literál* nebo *URI*. Zatímco URI má definovanou syntaxi podle příslušného RFC dokumentu literál nikoliv. Literálem může být řetězec, číslo, datum, adresa atd. Aby bylo možné literály korektně zpracovat, je vhodné literálu přiřadit datový typ. Datový typ literálu je nepovinný, ale pokud je datový typ znám je vhodné jej tímto způsobem definovat.

Pokud tedy chceme vyjádřit literál, který ponese datum a chceme, aby tento literál měl typ datum, potom k němu dodáme zdroj, kde je definována syntaxe a sémantika data, které jsme použili. Literál s typem se potom zapisuje obdobně jako samotný literál, jen se přidá



Obrázek 3.3: Příklad anonymního uzlu v RDF grafu.

URI, které je odděleno znaky "^^" od literálu. V př. 3.5 uvádím literál datum s příslušným datovým typem.

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.org/> .

ex:Petr foaf:birthday "1985-08-16"^^xsd:date .
```

Listing 3.5: Příklad literálu s typem.

3.6 Kontejnery

Kontejnery představuji prostředek pro popis skupiny věcí. Např. kniha má skupinu autorů nebo kurz má skupinu studentů atd. RDF definuje 3 typy kontejnerů.

Prvním typem kontejneru je *rdf:Bag*, který slouží k seskupeni zdrojů nebo literálů, kdy nám nezáleží na pořadí. Pokud je ovšem pořadí členů v kontejneru důležité použijeme kontejner typu *rdf:Seq*. Příkladem může být skupina slov, která musí být zpracována podle abecedy.

Třetím typem kontejneru je *rdf:Alt*, který představuje alternativní popis. Například máme originální název filmu a pomocí RDF chceme popsat také alternativní název v různých jazycích.

Jako příklad uvedu kontejner typu *rdf:Alt*, kde k originálnímu zdroji uvedu několik alternativních. Výrok: „RFC 2236 je možné nalézt také na http://tools.ietf.org/html/rfc2236, https://datatracker.ietf.org/doc/rfc2236/, http://www.hjp.at/doc/rfc/rfc2236.html" je v Turtle syntax zapsán v př. 3.6.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ex: <http://example.org/> .

<http://www.faqs.org/rfcs/rfc2236.html>  ex:alt-source  _:rfc2236
_:rfc2236  rdf:type  rdf:Alt
_:rfc2236  rdf:_1  <http://tools.ietf.org/html/rfc2236>
_:rfc2236  rdf:_2  <https://datatracker.ietf.org/doc/rfc2236/>
_:rfc2236  rdf:_3  <http://www.hjp.at/doc/rfc/rfc2236.html>
```

Listing 3.6: Ukázka alternativního kontejneru *rdf:Alt*.

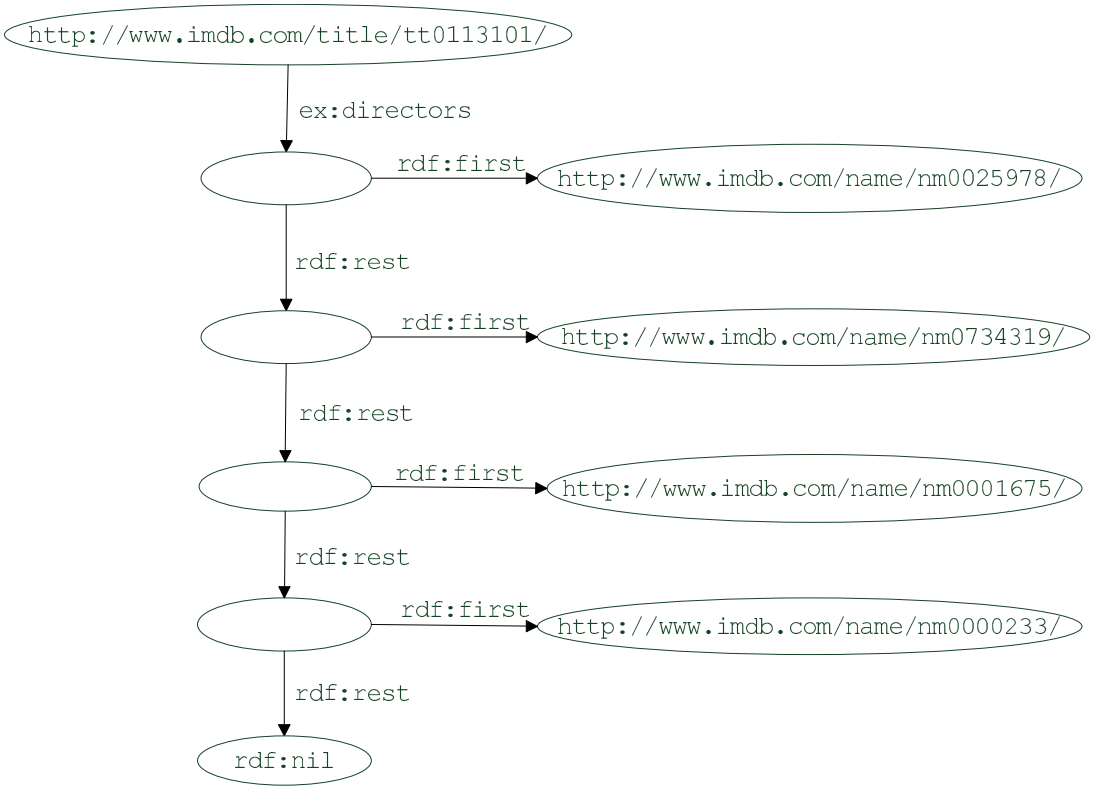
3.7 Kolekce

Kontejnery neumožňují určit pevný počet svých členů, vždy může existovat nějaký jiný zdroj nebo literál, jenž je členem kontejneru. Nikdy tak nemůžeme kontejner uzavřít. Pro vymezení pevného počtu členů existuje v RDF *kolekce*.

Kolekce je opět reprezentace skupiny věcí, narozdíl od kontejneru známe poslední prvek. Kolekce v RDF je velmi podobna seznamu v jazyku *Lisp*, máme vždy anonymní uzel, který ukazuje na první prvek seznamu a na zbytek seznamu.

Kolekce je vytvořena pomocí předdefinovaného typu *rdf:type* a předdefinovaných vlastností *rdf:first* a *rdf:rest* a předdefinovaného zdroje *rdf:nil*.

Mějme výrok obr. 3.4: „Four Rooms má režiséry Allison Anders, Alexandre Rockwell, Robert Rodriguez, Quentin Tarantino“. Výrok tedy říká, že nějaký zdroj má vlastnost režisér a přesně čtyři hodnoty této vlastnosti.



Obrázek 3.4: Příklad kolekce v RDF.

3.8 Serializace

RDF je pouze datovým modelem a pro publikaci, zpracování či výměnu dat je nutné datový model serializovat, tedy převést do textové podoby. RDF nabízí hned několik serializací, které se liší dle potřeby použití.

Nejběžnější a W3C doporučovanou serializací je serializace RDF do RDF/XML. RDF/XML je vhodné především pro strojové zpracování, díky velké podpoře XML v programovacích jazycích. V některých případech je vhodné použít zápis, který je bližší lidem a

tím je serializace N3(Notion 3), jejíž podmnožinou je Turtle(Terse RDF Triple Language), který je použit v této práci.

Populární je poslední dobou zápis pomocí RDFa (Resource Description Framework in attributes). Jde o sadu atributů, které rozšiřují jazyk XHTML o metadata. Prostřednictvím RDFa lze snadno přidat RDF metadata do současného webu. RDFa podporují indexovací roboti vyhledávačů Google[1] nebo Yahoo[2].

Pracovník společnosti Yahoo v oblasti Sémantického vyhledávání Peter Mika na svém weblogu[3] uvádí, že v roce 2010 RDFa používalo více než 430 milionů webových stránek, což je asi 3,6% všech webových stránek na internetu. Tento výzkum dále uvádí, že RDFa je nejrychleji rostoucí značkovací formát na WWW.

Pokud nasazení RDFa povede ke konkurenční výhodě, lze očekávat nárust nasazování této technologie i na dalších webových stránkách, což by pro Sémantický Web byl krok správným směrem.

RDFa se těší oblíbenosti především díky tomu, že jej lze snadno přidat do struktury již existujících stránek. Jako demonstraci RDFa uvádím př. 3.7 z RDFa Primer [6].

```
<div xmlns:dc="http://purl.org/dc/elements/1.1/">

   <div about="/alice/posts/trouble_with_bob">
      <h2 property="dc:title">The trouble with Bob</h2>
      <h3 property="dc:creator">Alice</h3>
      ...
   </div>

   <div about="/alice/posts/jos_barbecue">
      <h2 property="dc:title">Jo's Barbecue</h2>
      <h3 property="dc:creator">Eve</h3>
      ...
   </div>

   ...

</div>
```

Listing 3.7: Příklad RDFa v XHTML

[1] http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=99170
[2] http://developer.yahoo.com/blogs/ydn/posts/2008/09/searchmonkey_support_for_rdfa_enabled/
[3] http://tripletalk.wordpress.com/2011/01/25/rdfa-deployment-across-the-web/

Kapitola 4

SPARQL

Tato kapitola uvede čtenáře do dotazovacího jazyka SPARQL[4] a spolu s příklady vysvětlí jak SPARQL pracuje.

4.1 Úvod do SPARQLu

SPARQL(SPARQL Protocol and RDF Query Language) je dotazovací jazyk pro RDF. SPARQL byl 15.ledna 2008 standardizován organizací W3C. Nyní se pracuje na verzi 1.1[7], která řeší některé nedostatky a rozšiřuje současnou verzi z roku 2008.

Dotazovací jazyk SPARQL je „read-only“, tedy je možné s ním data jen získávat, nikoliv měnit nebo ukládat. Jazykem pro vkládání nebo změnu dat je jazyk SPARUL(SPARQL Update Language)[5], ten však nebyl standardizován a jedná se jen o návrh W3C, syntaxe je odvozena z jazyka SPARQL.

Jazyk SPARQL je založen na vyhledávání grafových vzorů(graph pattern). Nejjednodušším grafovým vzorem je vzor trojice(triple pattern). Vzor trojice je tvořen RDF trojicí, ovšem na místě subjektu, predikátu nebo objektu se vyskytuje proměnná, kterou hledáme. Kombinací vzorů trojic tak vytvoříme grafový vzor, který je vyhledán v RDF úložišti. Navázané proměnné jsou potom výsledkem dotazu.

Pro všechny příklady v této kapitole mějme definovaná RDF data př. 4.1:

```
@prefix ex: <http://example.com/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

ex:peter-sellers foaf:knows ex:john-walker .
ex:peter-sellers foaf:knows _:jane-hall .
_:jane-hall foaf:mbox <mailto:jane@example.org> .
_:jane-hall foaf:homepage <http://www.example.org/jane-hall> .
ex:john-walker foaf:mbox <mailto:j.walker@example.org> .
```

Listing 4.1: RDF data pro dotazování.

4.2 SPARQL dotaz

SPARQL je syntaktickým zápisem velmi podobný jazyku SQL a určitým konstrukcím, lze rozumnět i bez znalosti sémantiky syntaxe. Proměnné, které jsou výsledkem dotazu, jsou definovány v klauzuli *SELECT* a uvozují se znakem „?“ případně „$“. V klauzuli *FROM*

je specifikován zdroj dat, nad kterými se bude dotaz vyhodnocovat. V následující klauzuli *WHERE* je potom grafový vzor v němž jsou navázány proměnné z klauzule *SELECT*.

Nad definovanými RDF daty př. 4.1 provedeme dotaz. Vyhledáme adresy elektronické pošty osob, které zná Peter Sellers. Tento dotaz bude potom zapsán v SPARQL viz př. 4.2:

```
PREFIX ex: <http://example.com/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?mbox
WHERE {
  ex:peter-sellers foaf:knows ?someone
  ?someone foaf:mbox ?mbox
}
```

Listing 4.2: Dotaz v SPARQL.

Výsledek dotazu je potom v podobě tabulky 4.1. Vidíme, že byly nalezeny všechny objekty, které mají vlastnost *foaf:mbox* a zároveň subjekt této trojice je objektem s vlastností *foaf:knows* a subjektem *ex:peter-sellers* v jiné trojici.

mbox
<mailto:jane@example.org>
<mailto:j.walker@example.org>

Tabulka 4.1: Výsledek dotazu SPARQL

Speciálním případem je dotazování na shodu literálů. Literály jsou objekty RDF trojice a mohou mít *datový typ* nebo přiřazen štítek definující *přirozený jazyk* (lang tag). Pokud máme data, v nichž literál má definován štítek s přirozeným jazykem, potom aby došlo ke shodě, musí být v dotazu tento literál zapsán přesně s tímto štítkem. V případě boolean hodnot a čísel je datový typ rozpoznán při zpracování dotazu a je tedy nepovinný. Jiné datové typy musí být explicitně uvedeny.

4.3 Typy dotazů

SPARQL podporuje 4 typy dotazů, jsou to dotazy *SELECT*, *ASK*, *DESCRIBE* a *CONSTRUCT*. Rozdíl mezi dotazy je především ve výsledcích, které vracejí.

4.3.1 SELECT

Dotaz *SELECT* je analogií na zažitý dotaz SELECT v SQL. V klauzuli SELECT jsou definovány proměnné, které jsou navázány na grafový vzor v klauzuli WHERE. Vyhodnocování probíhá tak, že je nejdříve vyhledán grafový vzor a v případě shody jsou naplněny definované proměnné. Výsledkem dotazu SELECT je tabulka.

4.3.2 CONSTRUCT

Dotaz *CONSTRUCT* umožňuje vytvořit vlastní graf, jehož hodnoty jsou získány dotazem do RDF úložiště. Dotaz viz př. 4.3 se skládá z klauzule *CONSTRUCT*, ve které je definována šablona vytvářeného grafu. V klauzuli *WHERE* je grafový vzor, který je hledán včetně proměnných.

```
PREFIX ex <http://example.org/>
PREFIX foaf <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
  ?homepage ex:contact ?mail
}
WHERE {
  ?x foaf:mbox ?mail
  ?x foaf:homepage ?homepage
}
```

Listing 4.3: Dotaz CONSTRUCT v SPARQL.

Dotaz vrací RDF graf viz př. 4.4, přesně podle šablony:

```
<http://www.example.org/jane-hall> ex:contact <mailto:jane@example.org>
```

Listing 4.4: Výsledek dotazu SPARQL CONSTRUCT.

4.3.3 DESCRIBE

Dotaz *DESCRIBE* vyhledá všechny RDF trojice týkající se určitého subjektu. Pomocí tohoto dotazu lze tak snadno zjistit veškeré informace, kterém víme o daném subjektu.

Syntaxe je opět podobná předchozím typům dotazů. V klazuli DESCRIBE je deklarovaná proměnná a v následující klauzuli WHERE je grafový vzor s navázanou proměnnou z klauzule DESCRIBE.

4.3.4 ASK

Dotaz *ASK* se od předchozích liší v tom, že nevrací žádná RDF data, ale pouze *boolean* hodnotu. Ptáme se tak zda v úložišti existuje konkrétní grafový vzor. Dotaz je tvořen klauzulí ASK ve které je grafový vzor, na jehož existenci se dotazujeme.

4.4 Modifkátory výsledků dotazů

Vedle výše uvedených 4 typů dotazů existují další klauzule ovlivňující výslednou podobu vrácených dat.

4.4.1 LIMIT, OFFSET a ORDER BY

Klauzule *LIMIT*, *OFFSET* a *ORDER BY* mají stejnou sémantiku jako v SQL. Klauzulí *LIMIT* ovlivníme počet vrácených výsledku v dotazu. Tato klauzule je vhodná v situacích, kdy pokládámé příliš obecný dotaz a počet vrácených řádků je podstatně vyšší. Pro definici počátečního *offsetu* vrácených výsledků slouží klauzule *OFFSET*. Pomocí dvou výše uvedených klauzulí lze např. provádět stránkování výsledků dotazu.

SPARQL podporuje řazení výsledků a to pomocí klauzule *ORDER BY*. Řazení je stejně jako v SQL možné vzestupné či sestupné. SPARQL nedefinuje řazení všech možných RDF výrazů, např. literál a literál s typem, pro bližší specifikaci odkáži čtenáře na popis SPARQL [4] od W3C.

4.4.2 DISTINCT, FILTER a OPTIONAL

Klíčové slovo *DISTINCT* eliminuje opakující se řešení dotazu. Příkladem je řešení, které se naváže na tu samou proměnnou té samé RDF trojice jako další řešení, potom je takové řešení eliminováno.

Klíčovým slovem *FILTER* lze testovat, zda určitá proměnná nabývá konkrétní hodnoty nebo je v určitém intevalu hodnot. SPARQL potom definuje další klíčová slova jako *isURI*, *isLITERAL*, *isBLANK* pro testování hodnot proměnných.

V případě, kdy požadujeme zjistit např. adresu elektronické pošty osob a navíc i telefonní číslo, jehož neexistence by však neměla mít vliv na výsledek dotazu, použijeme klauzuli *OPTIONAL*. V této klauzli lze vyjádřit jakýkoliv volitelný grafový vzor, který pokud existuje, bude zahrnut ve výsledku dotazu.

Nad definovanými RDF daty viz př. 4.1 provedeme dotaz s klauzulí OPTIONAL. Vyhledáme adresy elektronické pošty osob, které zná Peter Sellers a navíc i volitelně domovské stránky těchto osob. Tento dotaz bude potom zapsán v SPARQL viz př. 4.5.

```
PREFIX ex <http://example.com/>
PREFIX foaf <http://xmlns.com/foaf/0.1/>
SELECT ?mbox ?homepage
WHERE {
  ex:peter-sellers foaf:knows ?someone
  ?someone foaf:mbox ?mbox
  OPTIONAL {
    ?someone foaf:homepage ?homepage
  }
}
```

Listing 4.5: Dotazu SPARQL s klauzulí OPTIONAL.

Ve výsledku dotazu tabulka 4.2 se potom objeví i RDF graf, kde vlastnost *foaf:homepage* není definována.

mbox	homepage
<mailto:jane@example.org> <mailto:j.walker@example.org>	<http://www.example.org/jane-hall >

Tabulka 4.2: Výsledek dotazu SPARQL s klauzulí OPTIONAL

Kapitola 5

Návrh RDF úložiště

5.1 Co je RDF úložiště?

RDF úložiště je datová základna, kde jsou uložena RDF data - v podstatě jde o systém řízení báze dat. Úložiště poskytuje přístup k uloženým datům a to obvykle prostřednictvím dotazovacího jazyka, v RDF je to nejčastěji SPARQL. Většina úložišť vedle dotazovacího jazyka umožňuje získat uložená RDF data jako výstupní soubor.

5.2 Specifikace požadavků

Pro řešení problému je nutné nedříve sjednotit požadavky. Cílem této práce je vytvoření knihovny pro ukládání a dotazování RDF dat. Implementačním jazykem knihovny má být PHP a data mají být uložena v relační databázi. Pro dotazování RDF dat má být implementována podmnožina vhodného dotazovacího jazyka.

5.3 Rozhranní úložiště

Rozhranní úložiště částečně vyplívá z požadavků, ale je nutné si veškeré rozhranní explicitně definovat.

Knihovna by měla poskytovat jednoduché rozhranní pro vložení jedné RDF trojice do úložiště. Půjde o metodu, která jako své argumenty ponese subjekt, predikát, objekt a volitelný typ literálu. Rozpoznání *URI* od *literálu* či zcela nevalidního vstupu bude ponecháno na samotné knihovně. Program, který bude využívat tuto knihovnu, může tímto rozhranním do RDF úložiště v cyklu vkládat neomezený počet RDF trojic z libovolného vstupu.

Vedle vkládání je důležitější a častěji prováděnou operací dotazovaní. První z metod pokrývajících získávání dat z úložiště je provedení dotazu ve vhodném dotazovacím jazyce. Této metodě bude předán jako argument řetězec s dotazem, který bude následně zpracován, přetransformován do SQL dotazu a ten bude vykonán v relační databázi.

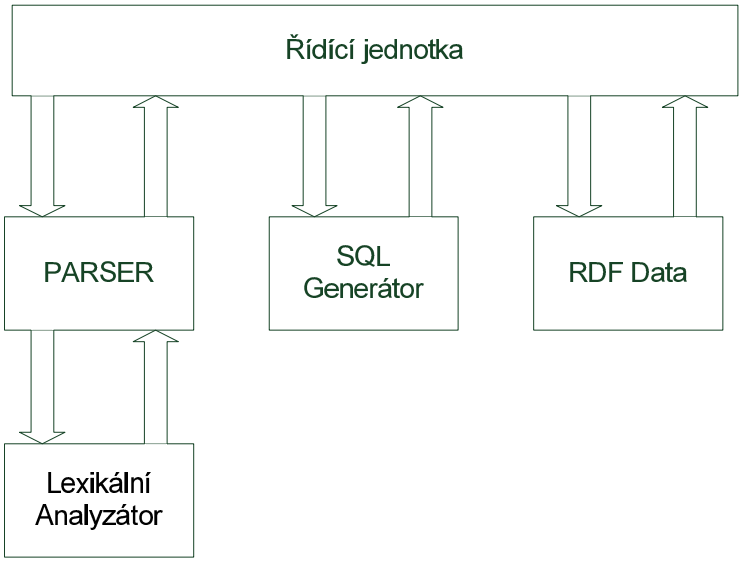
Druhou metodou pro dotazování RDF dat z úložiště bude implementačně jednodušší metoda, která najde všechny RDF trojice o určitém subjektu. Jde o velmi častý dotaz ekvivalentní dotazu *DESCRIBE* jazyka *SPARQL*, ale bez klauzule *WHERE*. Tato operace by měla být možná i v implementované podmnožině dotazovacího jazyka, ale jelikož jde o často prováděnou operaci, bylo vytvořeno toto speciální rozhraní díky němuž odpadá překlad SPARQL do SQL a výsledek je tudíž získan efektivněji.

Obě výše popsané metody, provedou dotaz v relační databázi a v případě neúspěchu vyhodí výjimku. Rozhranní pro získání dotázaných dat je tvořeno metodou vracející jeden řádek v asociativním poli a metodou vracející počet nalezených řádku. Pokud tak budeme chtít dostat celou tabulku, která bude výsledkem dotazu, voláme příslušnou metodu v cyklu.

5.4 Architektura úložiště

Úložiště je složeno z autonomních částí, které spolu tvoří požadovanou logiku. Architektura úložiště je vidět na obr. 5.1.

- Na nejvyšší úrovní se nachází **řídicí jednotka**, jejíž rozhranním je rozhranní celé knihovny. Tato řídící komponenta dále komunikuje s komponentami na nižší vrstvě a implementuje řídící logiku úložiště.
- **Lexikální analyzátor** analyzuje předložený řetězec a rozpoznává jednotlivé *tokeny*. Případně si ukládá jejich atributy např. načtený literál nebo jméno proměnné. Jádrem je konečný automat vhodné podmnožiny dotazovacího jazyka.
- Nad *lexikálním analyzátorem* je **parser**, který implementuje *ll gramatiku* a kontroluje syntaktickou a sémantickou korektnost dotazu. Vedle kontroly syntaxe a sémantiky zde dochází k zpracování dotazu do vhodných *datových struktur*.
- Generování *SQL dotazu* je implementováno samostatně v **SQL generátoru**. Tvorba *SQL dotazu* se tak děje až po zpracování dotazu a to na základě vytvořených datových struktur. Zde také dochází k sekudnární kontrole sémantiky dotazu.
- Poslední částí je komponenta **RDF data**, která zapouzdřuje přístup k relační databázi, v našem případě *MySQL*.



Obrázek 5.1: Architektura RDF uložiště.

5.5 Návrh relační databáze

Pokud chceme modelovat relační databázi pro RDF, potom se musíme zabývat tím, jak ukládat datový model EAV(entity-attribute-value) do relační databáze. Při návrhu relační databáze jsem si zvolil entity, které se vyskytují v RDF. Těmito entitami jsou *zdroje* a *literály*, které dohromady tvoří entity *trojice*. Entita trojice je ovšem pouhou abstrakcí, ve skutečnosti nenese žádná data a slouží jen ke spojení zdrojů a literálů.

5.5.1 Zdroj

Jak napovídá akronym RDF, *zdroje*(Resources) tvoří podstatnou část RDF trojic. Zdrojem může být v RDF trojici subjekt, predikát i objekt. Každý zdroj je reprezentován unikátním identifikátorem URI. Kromě tohoto identifikátoru nepotřebujeme pro entitu *zdroj* ukládat již další atributy. Zároveň také vytvoříme index pro atribut URI, neboť právě atribut URI se bude velmi často vyskytovat při dotazování v klazuli WHERE. Atribut URI je datového typu *VARCHAR* o velikosti 300 znaků.

5.5.2 Literál

Literál je v RDF objekt, který kromě své hodnoty může mít štítek definující přirozený jazyk v němž je literál zapsán. Implicitně je *literál* beztypový, ale lze mu typ dodefinovat. Pokud máme takto shrnuté požadavky, potom entita literál bude mít autributy *řetězec literálu*, který je typu *VARCHAR* o velikosti 99 znaků. Jak vyplinulo z testování, tato velikost je dostačující.

Druhým atributem je nepovinný atribut *lang tag*, jde o označení přirozeného jazyka v němž je literál vyjádřen.

Literál může mít také definován datový typ např. řetězec, celé číslo, datum atp. Tento datový typ je popsán pomocí zdroje. Proto má entita *literál* vztah k entitě *zdroj* N:1. V relační databázi bude mít potom libovolná n-tice relace *literál* cizí klíč, který pokud bude nenulový, tak bude identický s primárním klíčem některé n-tice relace *zdroj*.

5.5.3 Trojice

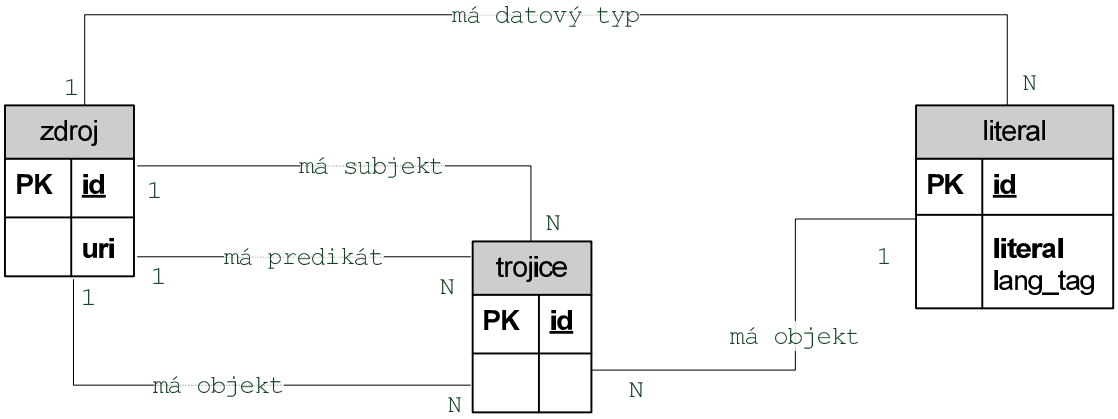
RDF trojice je tvořena *subjektem*, *predikátem* a *objektem*. *Objekt* může navíc být jak zdrojem tak i literálem. Prázdné uzly v této práci neuvažuji, ovšem pokud by byl zaveden nějaký jednoznačný idenitifkátor např. konkatenace řetezce „blank“ s primárním klíčem daného řádku, potom by bylo možné prázdné uzly do úložiště ukládat.

Entita *trojice* je tak tvořena vztahem *má subjekt* s entitou *zdroj*, *má predikát* s entitou *zdroj* a *má objekt* s entitou *zdroj* nebo s entitou *literál*.

Z toho vyplívá, že tabulka trojice je tvořena pouze 5 celými čísly, což by mělo usnadnit procházení této tabulky při vyhledávání:

- Primární klíč
- Povinný cizí klíč do tabulky zdroj - subjekt.
- Povinný cizí klíč do tabulky zdroj - predikát.
- Nepovinný cizí klíč do tabulky zdroj - objekt jako zdroj.
- Nepovinný cizí klíč do tabulky literál - objekt jako literál.

Výsledek návrhu relační databáze pro uložení datového modelu RDF lze vidět na ER diagramu obr. 5.2.



Obrázek 5.2: ER diagram RDF úložiště.

5.6 Vhodný dotazovací jazyk

Pro výběr vhodného dotazovacího jazyka jsem si zvoli několik kritérií. Vhodný dotazovací jazyk by neměl být příliš náročný na implementaci a měl by nabídnout jednoduchý a známý způsob tvorby ne příliš komplikovaných dotazů. Dotazem by mělo jít získat konkrétní subjekt, predikát nebo objekt, který vyhovuje určitému vzoru trojice.

Dotazovacích jazyků pro RDF je spousta možná proto, že ani jeden není zrovna dokonalý. Pro potřeby této práce jsem zvolil podmnožinu jazyka SPARQL, který je doporučením W3C pro dotazování nad RDF a lze jej považovat za standard v této oblasti.

Implementovaná podmnožina vychází z SPARQL a podporuje klauzule *PREFIX*, *SELECT*, *WHERE*, *OPTIONAL*, *LIMIT* a *OFFSET*. Příkladem dotazu, který používá výše uvedené klauzule a lze jej napsat v implementované podmnožině, je dotaz viz př. 5.1.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT $someuri $firstName ?lastName ?mbox ?phoneNum
WHERE {
  ?someuri rdf:type foaf:Person
  ?someuri foaf:firstName ?firstName
  ?someuri foaf:lastName ?lastName
  OPTIONAL {
    ?someuri foaf:mbox ?mbox
    ?someuri foaf:phone ?phoneNum
  }
}
LIMIT 10
OFFSET 20
```

Listing 5.1: Demonstrace možného dotazu ve zvoleném dotazovacím jazyku.

Všech šest výše uvedených klauzulí je *case-insensitive*, tedy všechny klauzule je možné zapisovat s malými i velkými písmeny či je zcela libovolně střídat. Zápis „Select“, „select“ nebo „SeLeCT“ je zcela legální v rámci implementovaného dotazovacího jazyka. Ostatní lexémy jsou už ovšem *case-sensitive*, proto např. prefix „rdf“ není totéž jako prefix „Rdf“.

Proměnné jsou uvozeny znakem „$“ nebo „?“, zároveň platí, že proměnná „$var“ je identická proměnné „?var“. Také platí, že jména proměnných jsou case-sensitive.

Implementovaná podmnožina nemá žádnou podporu slovníků, všechny URI jsou si rovnocenné a žádné URI nemá speciální význam. Také nejsou podporovány vestavěné funkce. Dále nebyly implementovány operátory(středník a čárka) pro zkrácený zápis trojic v grafovém vzoru. Několik hlavních odlišností od jazyka SPARQL je především v klauzuli WHERE.

V klauzuli *WHERE* lze definovat *grafové vzory* stejně jako v SPARQL s určitými omezeními:

- Nelze vnořovat jednotlivé grafové vzory.
- Není definována sémantika operátoru tečka mezi vzory trojice.
- Všechny proměnné, které jsou použity v klauzuli *WHERE* musí být deklarovány v klauzuli *SELECT*.

Klauzule *WHERE* dle definované gramatiky musí obsahovat alespoň jednu trojici. Trojice, která se nachází na 1. místě v klauzuli WHERE, označuji jako *první trojice*.

Pro *první trojici* v klauzuli WHERE platí:

- Musí mít libovolný *literál* nebo *URI*.
- Musí mít alespoň jednu *proměnnou*.
- Nesmí být *OPTIONAL*.

Aby bylo možné vytvořit grafový vzor, je nutné jednotlivé trojice v klauzuli *WHERE* propojit. To znamená, že jednotlivé trojice musí mít společný libovolný uzel. V tomto pohledu jsem byl při návrhu konzervativní a zvolil jsem propojení na základě proměnné v subjektu každé trojice. Tento způsob jsem zvolil, aby implementace obecného algoritmu pro překlad nebyla příliš složitá. Z toho vyplívá, že grafovým vzorem v implementované podmnožině lze popsat otcovský uzel a libovolný počet synovských uzlů. Lze tak popsat jednu úroveň v stromové struktuře.

Ostatní trojice v klauzuli WHERE musí být provázány s první trojicí a zároveň pro ně platí:

- Všechny trojice musí mít subjekt se stejnou proměnnou jako subjekt první trojice.
- Některé trojice mohou být *OPTIONAL*.
- Na místě predikátu nebo objektu musí být proměnná, ale ne na obou zároveň.

5.6.1 LL gramatika

Na základě LL gramatiky jazyka SPARQL, která je uvedena ve W3C doporučení pro jazyk SPARQL [4], jsem odvodil LL gramatiku implementované podmnožiny. Pro implementaci jsem také určitá pravidla, která definují konkrétní lexémy, z LL gramatiky vyňal a z nich vytvořil konečný automat.

Implementovaná podmnožina jazyka SPARQL je popsána *LL gramatikou* viz př. 5.2, kde neterminál je ohraničen znaky „<“ a „>“. Jednotlivé terminály jsou tvořeny *tokeny*, jenž jsou uvozeny znakem „t“. Jednotlivé *tokeny* jsou popsány konečným automatem obr. 5.3.

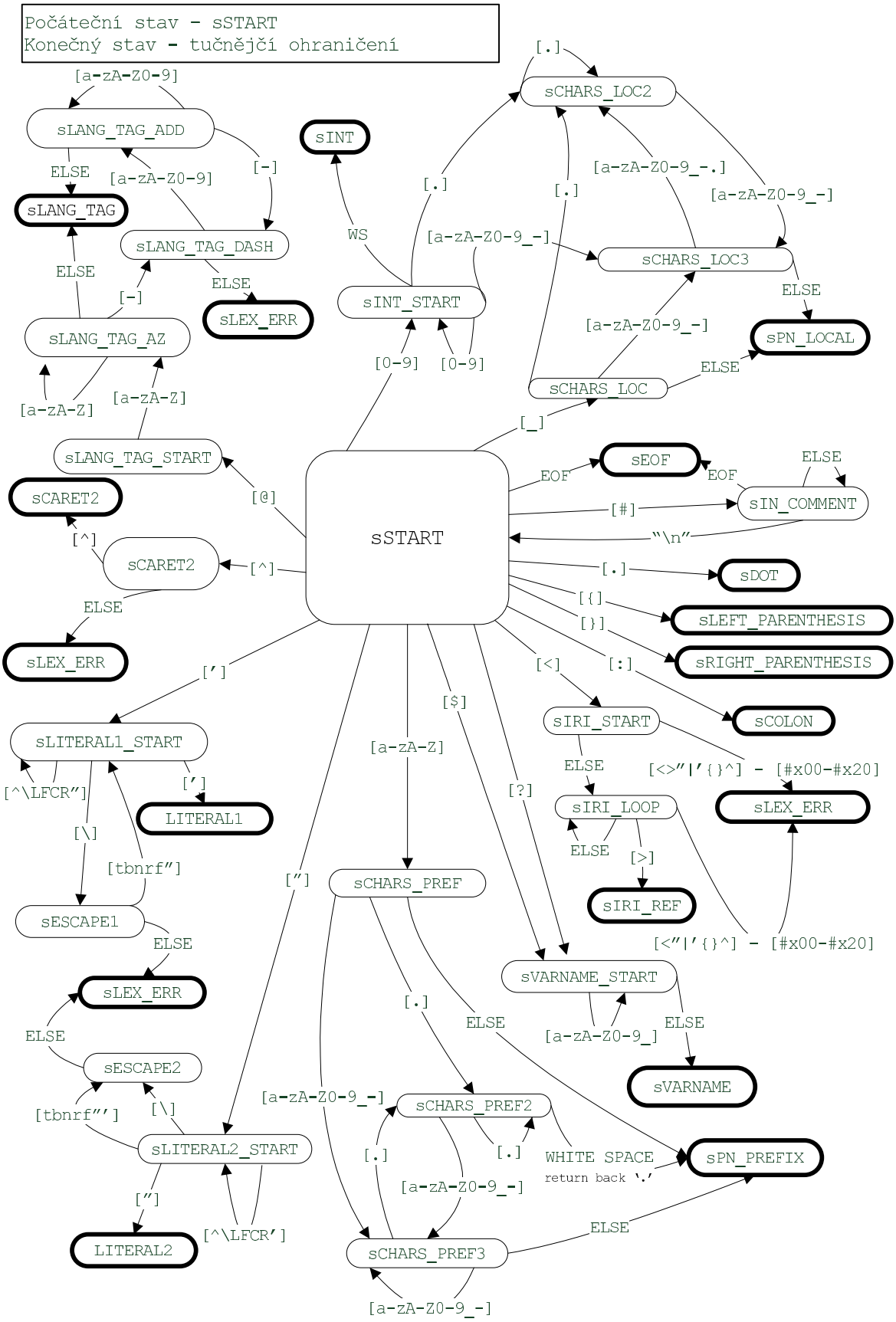
```
 1. <QUERY> -> <PROLOQUE> <SELECT_QUERY> tEOF
 2. <PROLOGUE> -> tPREFIX <PREFIX_DECLR> <PROLOGUE>
 3. <PROLOGUE> -> tSELECT
 4. <PREFIX_DECLR> -> tPN_PREFIX tCOLON tIRI_REF
 5. <SELECT_QUERY> -> <VAR_LIST> <WHERE_CLAUSE> <SOLUTION_MODIFIER>
 6. <VAR_LIST> -> tVARNAME <VAR_NEXT>
 7. <VAR_NEXT> -> tVARNAME <VAR_NEXT>
 8. <VAR_NEXT> -> tWHERE
 9. <WHERE_CLAUSE> -> tLEFT_PARENTHESIS <GRAPH_PATTERN>
10. <GRAPH_PATTERN> -> tOPTIONAL <OPTIONAL_GRAPH_PATTERN> <NEXT_GRAPH_PATTERN>
11. <GRAPH_PATTERN> -> <TRIPLET> <NEXT_GRAPH_PATTERN>
12. <NEXT_GRAPH_PATTERN> -> tRIGHT_PARENTHESIS
13. <NEXT_GRAPH_PATTERN> -> tOPTIONAL <OPTIONAL_GRAPH_PATTERN> <NEXT_GRAPH_PATTERN>
14. <NEXT_GRAPH_PATTERN> -> <GRAPH_PATTERN>
15. <OPTIONAL_GRAPH_PATTERN> -> tLEFT_PARENTHESIS <TRIPLET> <NEXT_OPTIONAL_TRIPLET>
16. <NEXT_OPTIONAL_TRIPLET> -> tRIGHT_PARENTHESIS
17. <NEXT_OPTIONAL_TRIPLET> -> <TRIPLET> <NEXT_OPTIONAL_TRIPLET>
18. <TRIPLET> -> <VAR_OR_IRIREF> <VAR_OR_IRIREF> <VAR_OR_IRIREF_OR_LITERAL> <
TRIPLET_DELIMITER>
19. <TRIPLET_DELIMITER> -> tDOT
20. <TRIPLET_DELIMITER> -> epsilon
21. <VAR_OR_IRIREF> -> tVARNAME
22. <VAR_OR_IRIREF> -> <IRI_REF>
23. <VAR_OR_IRIREF_OR_LITERAL> -> tVARNAME
24. <VAR_OR_IRIREF_OR_LITERAL> -> <LITERAL>
25. <VAR_OR_IRIREF_OR_LITERAL> -> <IRI_REF>
26. <LITERAL> -> tLITERAL <LITERAL_TYPE>
27. <LITERAL_TYPE> -> tLANGTAG
28. <LITERAL_TYPE> -> tLANGTAG tCARET2 <IRI_REF>
29. <LITERAL_TYPE> -> epsilon
30. <IRI_REF> -> tIRI_REF
31. <IRI_REF> -> tPN_PREFIX tCOLON <LOCAL_NAME>
32. <LOCAL_NAME> -> tPN_LOCAL
33. <LOCAL_NAME> -> tPN_PREFIX
34. <SOLUTION_MODIFIER> -> <LIMIT> <OFFSET>
35. <SOLUTION_MODIFIER> -> epsilon
36. <LIMIT> -> tLIMIT tINT
37. <OFFSET> -> tOFFSET tINT
38. <OFFSET> -> epsilon
```

Listing 5.2: LL gramatika dotazovacího jazyka.

5.6.2 Konečný automat

Konečný automat je vypočetní model, který může být v jednom z několika stavů, mezi kterými přechází na základě symbolu, které čte ze vstupu. Množina stavů musí být konečná a zároveň neprázdná. Konečný automat má jeden počáteční stav a konečnou, neprázdnou množinu koncových stavů. Konečný automat dále definuje tzv. *přechodové funkce*, které popisují pravidla přechodů mezi stavy.

Konečný automat v podobě grafu na obr. 5.3 má počáteční stav *sSTART* a koncové stavy jsou označeny tučnými ovály, které tak tvoří uzly grafu. Přechody jsou tvořeny orientovanými hrany grafu.



Obrázek 5.3: Konečný automat pro rozpoznání tokenů dotazovacího jazyka.

Kapitola 6

Popis implementace

V této kapitole je popsána implementace navrženého úložiště a to především překlad SPARQL na SQL. Implementačním jazykem je PHP(verze 5.2.13) a MySQL(verze 5.1.50).

6.1 Datové struktury

Především pro překlad dotazovacího jazyka na SQL je potřeba strukturovaně ukládat data z podmnožiny SPARQL dotazu do interních datových struktur. Tyto datové struktury jsou potom analyzovány při generování SQL dotazu. Při zpracování jsou vytvářeny datové struktury *VarTable*, *TripletTable* a dočasná datová struktura *PrefixTable* pro uložení prefixu a URI.

6.1.1 Datová struktura VarTable

Během překladu je potřeba uložit veškeré deklarované proměnné v klazuli *SELECT* a následně ověřit v klauzuli *WHERE*, že byly deklarované. Datová struktura *VarTable* vzniká v době překladu a je dostupná i po skončení zpracování dotazovacího jazyka.

Kromě samotného jména proměnné ukládáme do struktury unikátní jméno proměnné použité v SQL dotazu, nagenerovanou část SQL dotazu a zda-li je proměnná na pozici subjektu, predikátu či objektu.

6.1.2 Datová struktura TripletTable

TripletTable je datová struktura pro reprezentaci všech trojic v klauzuli *WHERE* a také hodnot v nepovinných klauzulích *LIMIT* a *OFFSET*.

Každá trojice může být volitelná nebo povinná a vždy je složena ze subjektu, predikátu a objektu. Subjekt, predikát nebo objekt potom mají svůj typ(*URI*, *promměná* nebo *literál*) a samotná data. Pokud je objekt literálem, potom může obsahovat položky definující přirozený jazyk(lang tag) nebo datový typ literálu.

6.1.3 Datová struktura PrefixTable

Na začátku dotazu před klauzulí *SELECT* lze definovat *prefix* jako jméno a jeho hodnotu. Tyto prefixy jsou potom užívány pro přehlednost v klauzuli *WHERE*.

Při deklaraci *prefixu* dojde k uložení *jména prefixu* a jeho *hodnoty* do tabulky *PrefixTable*. Pokud překladač narazí na daný prefix potom nahradí *jméno prefixu* jeho *hodnotou* z tabulky. Tabulka *PrefixTable* zaniká po skončení zpracování dotazovacího jazyka.

6.2 Zpracování dotazovacího jazyka

Zpracování dotazovacího jazyka probíhá ve třech vrstvách:

- Nad řetězcem je vytvořeno rozhranní pro čtení znak po znaku s ukončující konstantou *EOF*.
- Rozpoznání *tokenů*.
- Stavba derivačního stromu podle LL gramatiky a tvorba datových struktur pro následnou analýzu při tvorbě SQL dotazu.

6.2.1 Čtení znaků z řetězce

Na nejnižší vrstvě celého překladače pracuje čtení znaků z řetězce. Zde byla vytvořena určitá abstrakce čtení ze souboru se speciálním symbolem *EOF* a metodami *dej mi další znak* nebo *vrať načtený znak zpět*. V jazyce PHP verze 5 je určitá omezenost v podobě výpočtu délky řetězce v závislosti na kódování daného řetězce, proto v této práci uvažuji pouze anglickou abecedu bez diakritiky.

6.2.2 Lexikální analyzátor

Na vyšší vrstvě jsou jednotlivé znaky seskupovány do *tokenů*. Rozpoznávání *tokenů* je implementováno *konečným automatem*. Je implementována metoda *dej mi další token*, která začíná v počátečním stavu konečného automatu, čte jednotlivé znaky, dokud nedojde do konečného stavu. Vedle samotného tokenu zpřístupňuje i načtený *atribut*, který je již vhodně zpracován, např. *literál* nebo *URI* jsou zbaveny ohraničujích znaků *uvozovek* resp. *ostrých závorek*.

6.2.3 Parser

Na nejvyšší vrstvě dochází k stavbě derivačního stromu podle LL gramatiky, ověřování syntaxe a sémantiky jazyka a také k tvorbě datových struktur pro analýzu a tvorbu SQL dotazu.

Syntaktická analýza je implementována *rekurzivním sestupem*. Během *syntaktické analýzy* je částečně kontrolována sémantika, druhotná kontrola sémantiky probíhá při generování SQL dotazu. Vedle kontrol syntaxe a sémantiky probíhá také překlad, ovšem zatím jen tvorba datových struktur popsaných výše.

Pro ukázání základních principů si vzorový dotaz 5.1 rozdělíme na 4 části:

- Deklarace prefixů v klauzulích *PREFIX*.
- Deklarace proměnných v klauzuli *SELECT*.
- Trojice v klauzuli *WHERE*.
- Volitelné modifikátory *LIMIT* a *OFFSET*.

V první části jsou deklarovány unikátní jména prefixů a jejich hodnoty. Tyto nadeklarované dvojice jsou vkládány do tabulky *PrefixTable*. Pokud dojde k deklaraci již nadeklarovaného *prefixu* překladač potom ukončuje překlad a vrací vyjímku *sémantická chyba*.

V klauzuli *SELECT* jsou deklarovány proměnné, které se potom objeví ve výsledku a jejichž jména jsou vložena do tabulky *VarTable*. Opět není možné jednu proměnnou deklarovat dvakrát.

Datová struktura *TripletTable* je vytvářena v klauzuli *WHERE*, kde je grafový vzor skládající se z jedné či více trojic. Při zpracování jedné trojice pokud jde o proměnnou dochází k ověření, že byla deklarována, v případě prefixu je prefix nahrazen svou hodnou(URI) z tabulky *PrefixTable*. Po ověření je trojice vložena do datové struktury *TripletTable*. Postupně jsou takto do tabulky vkládány všechny trojice v klauzuli *WHERE*. Vedle trojic je do tabulky také vložena celočíselná informace v nepovinných klauzulích *LIMIT* a *OFFSET*.

6.3 Vytváření SQL dotazů

Největší výzvou celé práce byla algoritmizace vytváření SQL dotazu. Vytváření SQL dotazů implementuje komponenta *SQL generátor* viz. 5.1. Jde o komponentu, jenž je oddělena od komponenty pro *zpracování dotazovacího jazyka*. Můžeme tak říci, že překlad probíhá ve dvou fázích. Nejdříve je zpracován dotaz do datových struktur a následně jsou zpracovány datové struktury a vytvořen SQL dotaz. Tento postup mi tak umožnil rozdělit problém překladu na dílčí podproblémy. A to nejdříve se zaměřit na zpracování jazyka a až potom se věnovat transformaci na SQL dotaz.

Důležitým bodem pro automatické generování SQL dotazů je generování unikátních proměnných. Zde využívám zajímavé možnosti slabě typovaného jazyka *PHP*, který umožňuje inkrementaci znaků. Jazyk *PHP* operací *inkrementace* inkrementuje proměnnou nesoucí znak např. „a“ na „b“ a znak „z“ na „aa“ atd. Na tomto principu jsem vytvořil efektivní generátor proměnných, které budou použity v SQL dotazu.

6.3.1 Analýza SPARQL a SQL dotazů

Pro sestavení obecného algoritmu jsem nejdříve analyzoval podobu SQL dotazů. Vybral jsem lišící se typy dotazů z podomnožiny SPARQL a pro ně jsem ručně vytvořil SQL dotazy, které jsem otestoval zda vrací očekávané výsledky. Podobu SQL a SPARQL dotazů jsem analyzoval a hledal určité vzory, které se již nemění. Pro názornost tedy mějme jednoduchý dotaz v SPARQL př. 6.1. Dotazujeme se na proměnnou *someuri* a *age*. Tyto proměnné musí být navázané do grafového vzoru, kde *someuri* bude na místě subjektu a *age* na místě objektu a predikátem bude URI *http://xmlns.com/foaf/0.1/age*.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?someuri ?age
WHERE {
  ?someuri foaf:age ?age
}
LIMIT 10
OFFSET 20
```

Listing 6.1: Jednoduchý dotaz v SPARQL pro převod do SQL.

Z dotazu vyplívá, že vybíráme trojici, kde predikát má danou hodnotu a chceme znát jeho subjekt a objekt. Vytvořená knihovna z tohoto SPARQL dotazu vygeneruje SQL dotaz viz př. 6.2, který se na první pohled nejeví již tak dobře čitelný jako zápis v SPARQL.

```
SELECT
  CONCAT('<',a2.uri,'>') AS 'someuri',
  IFNULL(CONCAT('<',a4.uri,'>'),CONCAT('"', a5.literalstring, IFNULL(CONCAT('"^^<',
a6.uri,'>'), IFNULL(CONCAT('"@',a5.lang_tag),'"')))) AS 'age'
FROM triplet a1
  JOIN uri a2 ON a1.subject_id = a2.id
  JOIN uri a3 ON a1.predicate_id = a3.id
  LEFT JOIN uri a4 ON a1.object_iri_id = a4.id
  LEFT JOIN literal a5 ON a1.object_literal_id = a5.id
  LEFT JOIN uri a6 ON a5.uri_id = a6.id
WHERE a3.uri = 'http://xmlns.com/foaf/0.1/age'
LIMIT 10 OFFSET 20
```

Listing 6.2: Vygenerovaný dotaz SQL.

Při bližším pohledu vidíme, že se v klazuli *SELECT* dotazujeme na dvě proměnné jako v dotazu SPARQL. Jelikož proměnná *age* je v RDF trojici na místě objektu, může tak být URI nebo literálem. Jeden cizí klíč v tabulce *trojice* pro *objekt-literál* nebo *objekt-zdroj* je hodnoty *NULL*. Tento problém řeším klauzulí *IFNULL* jazyka *MySQL*, která umožňuje větvení toku programu. Potom v případě, kdy je *objekt-zdroj* hodnoty *NULL*, tak cizí klíč *objekt-literál* musí být platným cizím klíčem do tabulky *literál*. Podobně je řešen volitelný datový typ literálu nebo *lang tag*.

Druhou částí je klauzule *FROM*, kde vybíráme zdroj dat a tím je tabulka *trojice*. Tabulka *trojice* je spojena s ostatními tabulkami, pro spojení tabulek je použita klauzule *JOIN*. Všimněme si, že některé spjení nejsou povinné (cizí klíč může být hodnoty NULL), neboť RDF trojice nemusí mít objekt v tabulce URI, ale v tabulce literál, stejně tak datový typ literálu je nepovinný. Toto chování je implementováno klauzulí *LEFT JOIN*.

V klauzuli *WHERE* je potom podmínka říkající, že predikát dané trojice musí být *http://xmlns.com/foaf/0.1/age*. Klauzule *LIMIT* a *OFFSET* mají již stejnou sémantiku a syntaxi jako v SPARQL.

6.3.2 Sestavení obecného algoritmu

Z výše uvedeného příkladu je patrné, že určité vzory mapovaní SPARQL na SQL lze najít. Obecně platí, že počet proměnných v klauzuli *SELECT* v *SPARQL* se rovná počtu dotazovaných atributů v klauzuli *SELECT* v *SQL*. Pokud je proměnná v grafovém vzoru navázaná na objekt, potom je v dotazu *SELECT SQL* klauzule *IFNULL*, která vybere jako objekt *URI* nebo *literál*. Pro každou proměnnou tak musí být nagenerována část SQL dotazu, konkrétně část obsahu klauzule SELECT. Tato část je uložena do tabulky *VarTable* pro každou proměnnou. Existence části generovaného dotazu *SELECT* v tabulce *VarTable* je kontrolována na konci generování SQL dotazu. Pokud je nalezena proměnná, která není navázaná na žádný grafový vzor je vyhozena vyjímka *sémantická chyba*.

V grafovém vzoru se vedle proměnných vyskytují *literály* nebo *URI*. V tom případě jsou *literály* nebo *URI* použity v SQL klauzuli *WHERE* jako tzv. filtr podle kterého vybíráme trojice.

Důležité je propojení tabulek. Vždy vyhledáváme v tabulce *trojice*, která je spojena s tabulkou *zdroj* a *literál*. Toto spojení tabulek na základě hodnot *klíčů* je zapsáno klauzulí *JOIN*. V případech kdy cizí klíč *objekt jako zdroj* v tabulce *trojice* je nulové hodnoty(NULL), potom tato trojice má na místě objektu místo *zdroje literál* a spojení je řešeno klazulí *LEFT JOIN*.

Na základě výše uvedených principů byla implementována celá podmnožina dotazovacího jazyka, tak jak je definována v kapitole 5.6. Pro ukázku zde uvádím podobu vygene-

rovaného SQL dotazu př. 6.3, který odpovidá dotazu př. 5.1 z podmnožiny SPARQL, jenž byl uveden jako demonstrativní příklad dané podmnožiny dotazovacího jazyka.

```
SELECT
  CONCAT('<',a2.uri,'>') AS 'someuri',
  IFNULL(CONCAT('<',a9.uri,'>'),CONCAT('"', b0.literalstring, IFNULL(CONCAT('"^^<',
b1.uri,'>'), IFNULL(CONCAT('"@',b0.lang_tag),'"')))) AS 'firstName',
  IFNULL(CONCAT('<',b4.uri,'>'),CONCAT('"', b5.literalstring, IFNULL(CONCAT('"^^<',
b6.uri,'>'), IFNULL(CONCAT('"@',b5.lang_tag),'"')))) AS 'lastName',
  IFNULL(CONCAT('<',b9.uri,'>'),CONCAT('"', c0.literalstring, IFNULL(CONCAT('"^^<',
c1.uri,'>'), IFNULL(CONCAT('"@',c0.lang_tag),'"')))) AS 'mbox',
  IFNULL(CONCAT('<',c4.uri,'>'),CONCAT('"', c5.literalstring, IFNULL(CONCAT('"^^<',
c6.uri,'>'), IFNULL(CONCAT('"@',c5.lang_tag),'"')))) AS 'phoneNum'
FROM triplet a1
  JOIN uri a2 ON a1.subject_id = a2.id
  JOIN uri a3 ON a1.predicate_id = a3.id
  LEFT JOIN uri a4 ON a1.object_iri_id = a4.id
  LEFT JOIN literal a5 ON a1.object_literal_id = a5.id
  LEFT JOIN uri a6 ON a5.uri_id = a6.id
  JOIN triplet a7 ON a7.subject_id = a2.id
  JOIN uri a8 ON a7.predicate_id = a8.id
  LEFT JOIN uri a9 ON a7.object_iri_id = a9.id
  LEFT JOIN literal b0 ON a7.object_literal_id = b0.id
  LEFT JOIN uri b1 ON b0.uri_id = b1.id
  JOIN triplet b2 ON b2.subject_id = a2.id
  JOIN uri b3 ON b2.predicate_id = b3.id
  LEFT JOIN uri b4 ON b2.object_iri_id = b4.id
  LEFT JOIN literal b5 ON b2.object_literal_id = b5.id
  LEFT JOIN uri b6 ON b5.uri_id = b6.id
  LEFT JOIN triplet b7 ON (b7.subject_id = a2.id AND ( b7.predicate_id = ( SELECT b8
.id FROM uri b8 WHERE b8.uri = 'http://xmlns.com/foaf/0.1/mbox')))
  LEFT JOIN uri b9 ON b7.object_iri_id = b9.id
  LEFT JOIN literal c0 ON b7.object_literal_id = c0.id
  LEFT JOIN uri c1 ON c0.uri_id = c1.id
  LEFT JOIN triplet c2 ON (c2.subject_id = a2.id AND ( c2.predicate_id = ( SELECT c3
.id FROM uri c3 WHERE c3.uri = 'http://xmlns.com/foaf/0.1/phone')))
  LEFT JOIN uri c4 ON c2.object_iri_id = c4.id
  LEFT JOIN literal c5 ON c2.object_literal_id = c5.id
  LEFT JOIN uri c6 ON c5.uri_id = c6.id
WHERE a3.uri = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type' AND a4.uri = 'http
://xmlns.com/foaf/0.1/Person' AND a8.uri = 'http://xmlns.com/foaf/0.1/firstName' AND
 b3.uri = 'http://xmlns.com/foaf/0.1/lastName'
LIMIT 10
OFFSET 20
```

Listing 6.3: Demonstrace překladače: Vygenerovaný SQL dotaz pro vzorový SPARQL dotaz.

Kapitola 7

Testování RDF úložiště

Tato kapitola ukazuje výsledky testů, jenž byly provedeny na RDF úložišti. Jde o testování funkčnosti a výkonnosti.

7.1 Testování funkčnosti

Funkčnost RDF úložiště byla testována na menším vzorku dat - RDF graf čítající 26 trojic. Důležitým aspektem bylo vytvoření testů pokrývajících celou množinu všech dotazů, které jsou teoreticky možné v implementované podmnožině dotazovacího jazyka.

Během testování bylo nejdříve RDF úložiště prostřednictvím svého rozhrání naplněno RDF daty. Data byla vložena korektně a nedošlo k žádnému problému. Při vkládání bylo také ověřeno, že RDF úložiště je imunitní proti *SQL injection*, což je technika napadení databázové vrstvy programu vsunutím kódu přes neošetřený vstup.

Nad daty bylo provedeno 10 vybraných dotazů, které pokrývají vyjadřovací možnosti jazyka a výsledky vrácené těmito dotazy lze označit za korektní. Konkrétní testování s testovacími daty, dotazy a výstupy je uvedeno v příloze tohoto dokumentu v příslušné kapitole B.1.

7.2 Testování výkonnosti

U testování výkonnosti RDF úložiště měřím průměrnou dobu potřebnou pro vložení jedné trojice do relační databáze a dobu potřebnou pro vykonání dotazu. Výkonnost dotazování testuji jak výkonnost překladače, tak výkonnost MySQL úložiště. Zjišťuji tak dobu potřebnou pro překlad SPARQL na SQL a dobu potřebnou pro vykonání SQL dotazu v relační databázi.

Pro testování je nutné získat dostatečné množství dat, nad kterými bude možné vykonat smysluplné dotazování. Pro tyto účely jsem použil použil předpřipravená geografická data US Kongresu v N3 formátu z webu rdfabout.com[1]. Současné RDF úložiště nemá implementovánu žádnou podporu pro čtení dat v N3 formátu, proto jsem použil již existující PHP knihovnu ARC2[2] pro parsování vstupních trojic a potom už rozhranní knihovny RDF úložiště pro vložení trojice.

Důležité je zmínit, jak probíhá měření výkonnosti. Měřím dobu překladu, přesněji dobu od vytvoření instance třídy pro zpracování dotazu, volání metody pro zpracování dotazu

[1] http://www.rdfabout.com/demo/census/
[2] https://github.com/semsol/arc2/wiki

do interních datových struktur, vytvoření instance třídy pro generování SQL dotazu a vygenerování SQL dotazu na základě datových struktur. Dále je měřena doba potřebná pro vykonání SQL dotazu v relační databázi a to od vytvoření instance až po provedení dotazu. Pří vkládání jedné trojice měřím dobu začínající voláním metody a končící posledním příkazem této metody.

Výkonnost RDF úložiště, byla měřena na 4 vzorcích dat(A,B,C,D) o počtu stovek(737), tisíců(4807), desetitisíců(41 847) a statisíců(176 291) RDF trojic. Testování probíhalo na školním serveru eva.fit.vutbr.cz[3].

Testování proběhlo na 4 různých vzorcích dat, tedy struktura dat byla pokaždé mírně odlišná. Proto všechny dotazy nemohly být zcela identické, z toho důvodu byly vytvořeny vzory dotazů, podle nichž byly vytvořeny konkrétní dotazy pro konkrétní data. Vytvořil jsem 9 vzorů dotazů, podle nichž jsem pro konkrétní struturu dat navrhl konkrétní dotaz. Desátým dotazem bylo testování rozhrání pro výpis všech trojic o určitém subjektu. Kompletní seznam testů a jejich výstupů je uveden v příloze tohoto dokumentu v příslušné kapitole B.2. Dále uvádím jen výsledky testu.

7.2.1 Vkládání trojic

Pří vkládání dat do RDF úložiště je dodržena integrita dat. Při vložení jedné trojice se kontroluje, zda URI nebo literál, jež tvoří trojici, se již nevyskytuje v příslušné tabulce. V případě, že již v tabulce existují, tak se znovu nevkládají. Tato režie je hlavním faktorem, který zpomaluje vkládání trojic, na druhou stranu vkládání je druhotnou operaci. Důležitější a častěji prováděnou operací je dotazování, při kterém se tak nemusí procházet duplicitní řádky a je tudíž rychlejší.

Před samotným dotazováním čtyř sad dat bylo nejdříve potřeba tyto data vložit do RDF úložiště. V tabulce 7.1 lze vidět průměrný čas, který byl potřeba na vložení jedné trojice a jak jsem zmínil výše s větším množstvím trojic roste také potřebný čas na vložení jedné trojice.

Data	Počet trojic	Průměrná doba vložení jedné trojice [s]
A	737	0,0017
B	4 807	0,0024
C	41 847	0,0101
D	176 291	0,0304

Tabulka 7.1: Výsledky výkonnostních testů pro vkládání trojic do RDF úložiště

Sady dat *C* a *D* jsem rozdělil na několik souborů protože skript vkládající tak rosáhlá data by přesáhl maximální dobu po kterou může jeden PHP skript na serveru běžet a také jsem neměl přístup pro změnu nastavení této konfigurace. Díky tomu tak pro sadu dat *D*, která čítá 176 291 trojic, mám dílčí statistické údaje o vkládání, které jsou zobrazeny v tabulce 7.2. V této tabulce je v prvním sloupci zaznamenán počet již vložených trojic, v druhém sloupci počet trojic, které budou vloženy a z nichž se počítá průměrná doba na vložení jedné trojice. Průměrná doba potřebná na vložení jedné trojice není počítána z

[3]SW: FreeBSD 8.x, sendmail, NFS, Apache, Samba, IMAP4. HW: SuperMicro 6025B-tr+V, MB X7DWN+, 2xIntel Xeon 5420 (Core2 Quad 2,5GHz/8MB), 8 GB RAM, 150 GB HDD + 5 TB RAID-6 LSI 8888ELP/Promise VTrak J310s

celkového počtu již vložených trojic, ale právě z počtu, který je vynesen v druhém sloupci tabulky. Proto můžeme vidět jak se tento údaj měnil v průběhu vkládání.

Během testování bylo vloženo přes 200 000 RDF trojic. Nešlo o žádná náhodná čísla, nýbrž o reálná geografická data. Byl ověřen návrh tabulek relační databáze zejména velikost atributů *URI* a *literal*, která byla zcela dostačující. Na základě tohoto testování lze tvrdit, že byla ověřena také funkčnost neboť všechny trojice byly vloženy korektně.

Celkem trojic	Vložené trojice	Prům. doba na vložení 1 trojice [s]
0	15037	0,004362
15037	12177	0,009718
27214	10109	0,014112
37323	10845	0,016599
48168	12453	0,019532
60621	9680	0,021935
70301	8855	0,026321
79156	8497	0,029509
87653	7215	0,030012
94868	9527	0,033167
104393	8085	0,037800
112480	8976	0,040631
121456	7700	0,041682
129156	7612	0,043735
136768	6489	0,046741
143257	8504	0,048591
151761	8294	0,052897
160055	8712	0,055036
168767	7524	0,057046

Tabulka 7.2: Statistika při vkládání RDF dat D

7.2.2 Dotazování

Testovací dotazy, které jsem provedl nad RDF úložištěm, lze rozdělit na 4 skupiny:

- Čtyři dotazy I. mající jako grafový vzor pouze jednu trojici.
- Tři dotazy II. mající jako grafový vzor dvě trojice.
- Dva dotazy III. mající jako grafový vzor čtyři trojice.
- Jeden dotaz IV. na popis určitého zdroje. Zde není počítán SQL dotaz.

Testování výkonnosti bylo prováděno automatizovaně, kdy program pro sadu dat *A*, *B* a *C* provedl 100 krát pro data *D* 20 krát připravené dotazy a následně ze získaných časů spočítal průměr. Doba překladu pro jednotlivé dotazy a data je v tabulce 7.3.

Tabulka tak ověřuje, že dotazy, jež byly vytvořeny pro každou sadu dat zvlášť na základě určitého vzoru, se příliš neliší, neboť průměrný čas překladu je přibližně podobný.

Nejdéle trvaly překlady dotazů *III.a* a *III.b*, které v klauzuli WHERE obsahovaly složitější grafový vzor skládající se z 4 trojic.

dotaz	data A	data B	data C	data D
I.a	0,001402 s	0,001300 s	0,001406 s	0,001700 s
I.b	0,001414 s	0,000999 s	0,000995 s	0,001005 s
I.c	0,001408 s	0,001190 s	0,001202 s	0,001305 s
I.d	0,001907 s	0,001400 s	0,001409 s	0,001420 s
II.a	0,002202 s	0,001801 s	0,001808 s	0,001810 s
II.b	0,002339 s	0,001900 s	0,001909 s	0,002095 s
II.c	0,002249 s	0,002001 s	0,001807 s	0,002000 s
III.a	0,003506 s	0,003501 s	0,003504 s	0,003525 s
III.b	0,003404 s	0,003401 s	0,003409 s	0,003655 s
IV.	0,000000 s	0,000000 s	0,000000 s	0,000000 s

Tabulka 7.3: Výkonnost překladače RDF úložiště pro jednotlivé dotazy

V tabulce 7.4 je doba potřebná pro vykonání SQL dotazu v relační databázi. Z tabulky vyplívá, že některé dotazy mají průměrnou dobu pro vykonání SQL dotazu velmi podobnou pro všechny objemy dat, těmi dotazy jsou *I.a*, *I.d* případně i *I.c* *II.c* nebo *IV.*. Tyto dotazy jsou velmi konkrétní a nedochází tak procházení celých tabulek, narozdíl od ostatních dotazů, které jsou natolik obecné, že ve výsledku mohou mít až několik stovek či tisíc řádků.

dotaz	data A	data B	data C	data D
I.a	0,000249 s	0,000212 s	0,000241 s	0,000260 s
I.b	0,000254 s	0,000342 s	0,001116 s	0,390940 s
I.c	0,000297 s	0,000242 s	0,000231 s	0,000530 s
I.d	0,000235 s	0,000208 s	0,000210 s	0,000240 s
II.a	0,000272 s	0,000401 s	0,001529 s	0,980820 s
II.b	0,000269 s	0,000400 s	0,001409 s	1,061770 s
II.c	0,000220 s	0,000203 s	0,000237 s	0,000535 s
III.a	0,000303 s	0,000460 s	0,002008 s	2,195895 s
III.b	0,000308 s	0,000422 s	0,001710 s	2,032350 s
IV.	0,000214 s	0,000218 s	0,000245 s	0,000575 s

Tabulka 7.4: Výkonnost SQL enginu RDF úložiště pro jednotlivé dotazy

Kapitola 8

Závěr

Během této práce jsem vytvořil všeobecně použitelnou knihovna pro ukládání a dotazování dat ve fromátu RDF. Úspěšně jsem implementoval podmnožinu dotazovacího jazyka SPARQL. Tuto knihovnu jsem následně řádně otestoval a jednotlivé testy ukázali na vhodnost návrhu relační databáze i architektury RDF úložiště.

V otázce dalšího vývoje je nutné provést srovnání s již existujícími knihovnami podobného zaměření a zhodnotit, zda má smysl další vývoj tohoto úložiště a případně jakým směrem se ubírat.

Jako jedna z možností dalšího vývoje se nabízí implementace softwarového agenta, který by procházel web a plnil úložiště RDF daty. Následně by bylo vhodné data z tohoto úložiště zpřístupnit dalším programům např. pomocí protokolu SOAP jako Web Service. V takovém případě je nutné se zabývat autentizací uživatelů, zajímat se o bezpečnost a optimalizaci výkonu, především „kešování“ výsledků častých dotazů a překladu PHP např. APC(Alternative PHP Cache).

Při této práci jsem získal povědomí o pojmu Sémantický Web a seznámil se s technologiemi RDF a SPARQL. Významným přínosem pro mě bylo získání nových zkušeností a prohloubení znalostí jazyků PHP a MySQL. Odvětví Sémantické Webu vnímám jako možnou cestu vývoje současné podoby webu se spoustou potenciálu pro vývojáře webových aplikací.

Literatura

[1] Antoniou, G.; van Harmelen, F.: *A Semantic Web Primer, second edition*. The MIT Press Cambridge, 2008, iSBN 978-0-262-01242-3.

[2] Tim Berners-Lee, J. H.; Lassila, O.: The Semantic Web. `http://www.scientificamerican.com/article.cfm?id=the-semantic-web`, 2001-05-17 [cit. 2011-04-16].

[3] W3C: RDF Primer. `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`, 2004-02-10 [cit. 2011-04-16].

[4] W3C: SPARQL Query Language for RDF. `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`, 2008-01-15 [cit. 2011-04-16].

[5] W3C: SPARQL Update. `http://www.w3.org/Submission/2008/SUBM-SPARQL-Update-20080715/`, 2008-07-15 [cit. 2011-04-16].

[6] W3C: RDFa Primer. `http://www.w3.org/TR/2008/NOTE-xhtml-rdfa-primer-20081014/`, 2008-10-14 [cit. 2011-04-18].

[7] W3C: SPARQL 1.1 Query Language. `http://www.w3.org/TR/2010/WD-sparql11-query-20101014/`, 2010-10-14 [cit. 2011-04-16].

Dodatek A

Příklad použití knihovny RDF úložiště

Nejdříve je potřeba vložit zdrojové kódy, stačí vložit soubor `rdfrepository.php`, ve kterém je definována třída `RDFrepository`. Ostatní zdrojové kódy jsou již vkládány automaticky. Vždy je nutné vytvořit instanci třídy `RDFrepository`, třída nemá žádnou statickou metodu.

Metody knihovny `RDFrepository` s modifikátorem přístupu *public*:

- `insert_triplet()` vloží jednu RDF trojici do RDF úložiště
- `query()` vykoná dotaz v RDF úložišti
- `describe_subject()` dotáže všechny trojice určitého subjektu v RDF úložišti
- `num_rows()` vrátí počet řádků, jichž bylo dosaženo při dotazu
- `fetch_assoc()` vrátí jeden řádek výsledku dotazu
- `get_sql_query()` zpřístupňuje sql dotaz, který je výsledkem překladu, tato metoda je vhodná pro ladění nebo jako demonstrace funkčnosti překladače

Povinné parametry konstruktoru třídy RDFrepository:

- **Host** - identifikace počítače na němž běží MySQL server
- **User** - uživatelské jméno pro přihlášení k MySQL serveru
- **Password** - heslo pro přihlášení k MySQL serveru
- **Database name** - jméno databáze ve které se nachází příslušné tabulky a data

A.1 Vkládání RDF trojice

RDF úložiště umožňuje vložit jednu trojici skládající se ze *subjektu*, *predikátu*, *objektu* případně *datového typu* objektu nebo *štítku jazyka*(language tag). Vložení RDF trojice je vykonáno metodou *insert_triplet*, které jsou předány jako argumenty *subjekt*, *predikát* a *objekt*, volitelným parametrem metody je *datový typ* objektu nebo *jazykový štítek*.

Při vkládání trojice může dojít k vyjímce např. v případě již existující trojice. Dále dochází k ověření správnosti formátu a délky vkládaných dat. Kód PHP pro vkládání trojice viz př. A.1.

Parametry metody `insert_triplet`:

- Povinný parametr **subjekt**, je očekáváno URI.
- Povinný parametr **predikát**, je očekáváno URI.
- Povinný parametr **objekt**, pokud není URI je s ním nakládáno jako s literálem.
- Volitelný parametr **typ objektu**, může být datovým typem(URI) nebo jazykovým štítkem(language tag).

```
// pripojeni zdrojovych kodu
require_once 'rdfrepository.php';

// definice rdf dat
$sub = 'http://example.org/article/2641';
$pred = 'http://purl.org/dc/elements/1.1/title';
$obj = 'What did RDF mean in World War 2?';
$obj_type = NULL;

// vytvoreni instance tridy
$rdf_repository = new RDFrepository('host','user','password','dbname');

try {
  // vlozeni rdf dat do uloziste
  $rdf_repository->insert_triplet($sub, $pred, $obj, empty($obj_type)?'':$obj_type);
} catch (Exception $e) {
  // osetreni vyjimky
  echo $e->getMessage();
}
```

Listing A.1: Vložení RDF trojice prostřednictvím knihovny RDFrepository.

A.2 Vykonání dotazu

Metoda *query* vykoná dotaz z implementované podmnožiny dotazovacího jazyka SPARQL. V případě neúspěchu překladu nebo dotazu je vyhozena vyjímka.

```
// vlozeni knihovny rdf uloziste
require_once 'rdfrepository.php';

// dotaz
$query = 'SELECT .....';

// vytvoreni instance objektu
$rdf_repository = new RDFrepository('host', 'user', 'password', 'dbname');

try {
  // provedeni dotazu
  $rdf_repository->query($query);
} catch (Exception $e) {
  // osetreni vyjimky
  echo $e->getMessage();
}
```

Listing A.2: Provedení dotazu prostřednictvím knihovny RDFrepository.

A.3 Dotaz na jeden zdroj

Častým dotazem je získání všech RDF trojic o konkrétním subjektu. Pro tento účel je v RDF úložišti implementovaná metoda *describe_subject()*, která jako parametr přebírá *subjekt*, jež bude popsán.

```
// vlozeni knihovny rdf uloziste
require_once 'rdfrepository.php';

// rdf subjekt
$subject = 'http://example.com/';

// vytvoreni instance objektu
$rdf_repository = new RDFrepository('host', 'user', 'password', 'dbname');

try {
  // dotazani se na trojice subjektu
  $rdf_repository->describe_subject($subj);
} catch (Exception $e) {
  // osetreni vyjimky
  echo $e->getMessage();
}
```

Listing A.3: Popsání RDF subjektu prostřednictvím knihovny RDFrepository.

A.4 Získání dotázaných dat

Po provedeni dotazu lze ověřit počet řádků, jež byly dotazem získány a to prostřednictvím metody *num_rows()*. Jeden řádek je získán metodou *fetch_assoc()* jako asociativní pole, pro vypsání všech získaných dat je tato metoda volána v cyklu. Příklad A.4 ukazuje ověření počtu získaných výsledku i získání dat.

```
...
...
...

// ziskani dotazanych dat
if($rdf_repository->num_rows > 0) {
  while($row = $rdf_repository->fetch_assoc()) {
    // vypsani jednoho ziskaneho radku
    foreach ($row as $name=>$data) {
      echo $name; // nazev navazane promenne
      echo $data;  // dotazana data
    }
  }
}
else {
  // zadny vysledek
  echo 'Pro dany dotaz nebyl nalezen zadny vysledek';
}
```

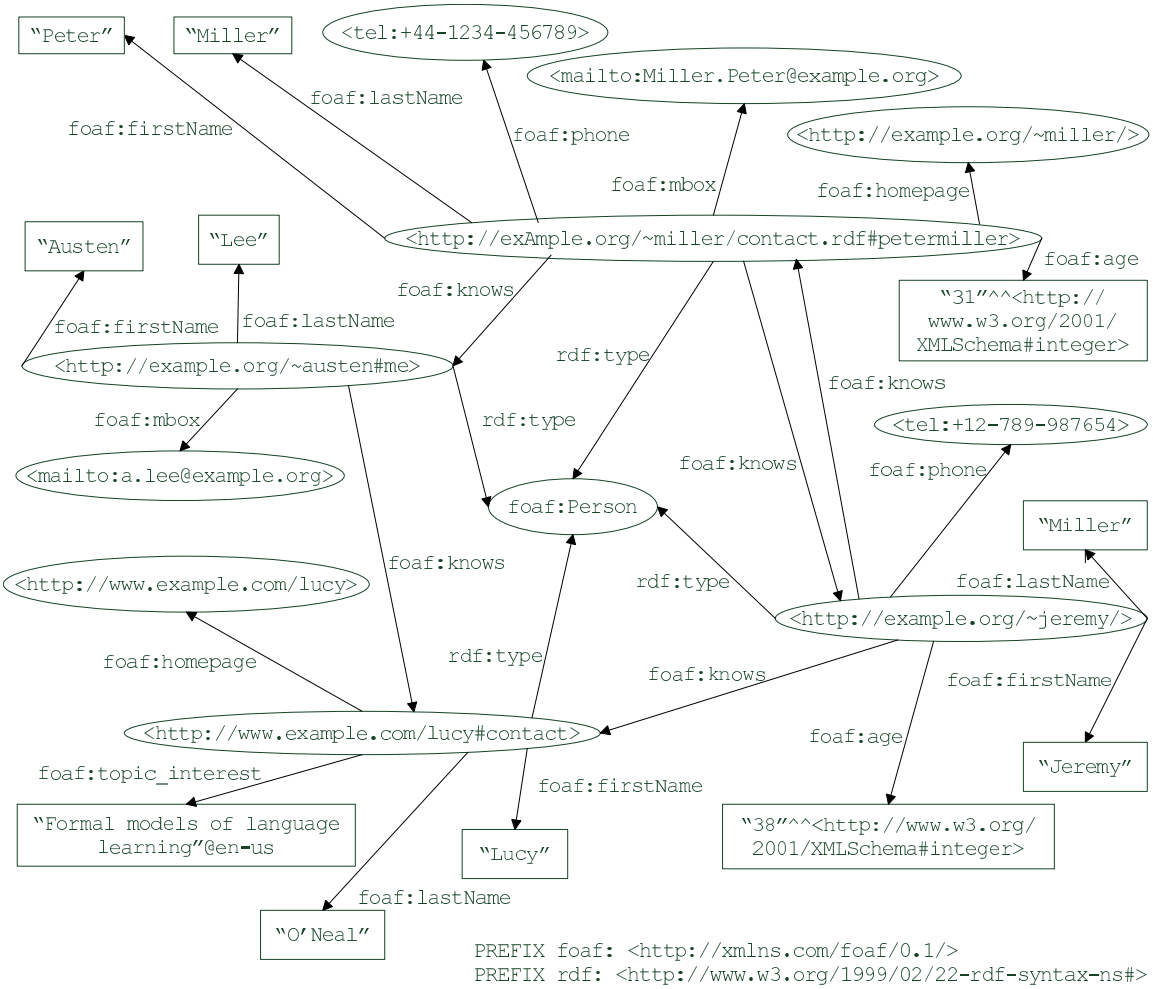
Listing A.4: Získání dotázaných dat.

Dodatek B

Testy provedené na RDF úložišti

B.1 Testy ověřující funkčnost

Pro ověřování funkčnosti byl vytvořen RDF graf obr. B.1, který pokrývá veškerou problematiku, jenž chceme testovat.



Obrázek B.1: Testovací RDF data.

Testovaní se skládá z deseti vybraných dotazů, kdy každý dotaz testuje odlišné vlastnosti funkčnosti.

B.1.1 Dotaz 1

Jednoduchý dotaz př. B.1, který testuje vyhledání proměnné *firstName*, která je navázána na objekt v grafovém vzoru, jež je tvořen jednou trojicí.

```
SELECT ?firstName
WHERE
{
  <http://example.org/~jeremy/> <http://xmlns.com/foaf/0.1/firstName> ?firstName .
}
```

Listing B.1: Jednoduchý dotaz na objekt.

firstName
„Jeremy“

Tabulka B.1: Výsledek dotazu 1

B.1.2 Dotaz 2

Ověření funkčnosti klauzule *PREFIX* př. B.2, která umožňuje definování zkráceného zápisu jmenného prostoru a následné použití v grafovém vozoru.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?firstName
WHERE
{
  <http://example.org/~jeremy/> foaf:firstName ?firstName .
}
```

Listing B.2: Jednoduchý dotaz na objekt s užitím klauzule PREFIX.

firstName
„Jeremy“

Tabulka B.2: Výsledek dotazu 2

B.1.3 Dotaz 3

Dvě proměnné v grafovém vzoru znamenají obecnější dotaz a tudíž i více vrácených řádků výsledku. Dotazujeme se na subjekt a objekt, jež mezi sebou mají vlastnost *firstName*. Dotaz viz př. B.3 a jeho výsledek tabulka B.3.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?uri ?firstName
WHERE
{
  ?uri foaf:firstName ?firstName .
}
```

Listing B.3: Dotaz s dvěma proměnnými.

uri	firstName
<http://example.org/~miller/contact.rdf#petermiller>	„Peter“
<http://example.org/~austen#me>	„Austen“
<http://www.example.com/lucy#contact>	„Lucy“
<http://example.org/~jeremy/>	„Jeremy“

Tabulka B.3: Výsledek dotazu 3

B.1.4 Dotaz 4

Dalším testovacím dotazem je ověření funkčnosti dotazu, kde je grafový vzor tvořen dvěma trojicemi. V dotazu př. B.4 se tážeme na URI, které je typu *osoba* a má adresu elektronické pošty, tuto adresu také chceme dotazem získat. Ve výsledku viz tabulka B.4 jsou potom jen ty URI, které mají vlastnost *mbox* a hodnota této vlastnosti.

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?uri ?mbox
WHERE {
  ?uri rdf:type foaf:Person
  ?uri foaf:mbox ?mbox
}
```

Listing B.4: Grafový vzor s dvěma trojicemi.

uri	mbox
<http://example.org/~miller/contact.rdf#petermiller>	<mailto:Miller.Peter@example.org>
<http://example.org/~austen#me>	<mailto:a.lee@example.org>

Tabulka B.4: Výsledek dotazu 4

B.1.5 Dotaz 5

Pokud se dotazujeme na literál, je podstatné, zda uvedeme jeho jazykový štítek(language tag). V testovacích datech má literál, jež budeme dotazovat, definován jazykový štítek. V dotazu viz př. B.5 však uvedeme pouze literál bez dalších vlastností, dle W3C specifikace pro SPARQL je očekáváno chování, že nedojde ke shodě. Toto chování platí i pro implementované úložiště. Výsledek viz tabulka B.5 dotazu je tak znázorněn jako prázdná tabulka.

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?uri ?firstName
WHERE {
 ?uri foaf:topic_interest "Formal models of language learning"
 ?uri foaf:firstName ?firstName
}
```

Listing B.5: Dotaz na shodu literálu bez jazykového štítku.

uri	firstName

Tabulka B.5: Výsledek dotazu 5

B.1.6 Dotaz 6

V následujícím dotazu vyjdeme z předchozího dotazu viz př. B.5 a literálu přidáme jazykový štítek *en-us*. V tomto případě př. B.6 tak dojde ke shodě a výsledkem je neprázdná tabulka B.6.

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?uri ?firstName
WHERE {
 ?uri foaf:topic_interest "Formal models of language learning"@en-us
 ?uri foaf:firstName ?firstName
}
```

Listing B.6: Dotaz na shodu literálu s jazykovým štítkem.

uri	firstName
<http://www.example.com/lucy#contact>	„Lucy“

Tabulka B.6: Výsledek dotazu 6

B.1.7 Dotaz 7

Dotaz 7 testuje shodu na literál bez datového typu. Narozdíl od SPARQL implementovaná podmnožina zapisuje i čísla do uvozovek a nedokáže rozpoznat číslo od řetězce znaků. Z tabulky B.7 s výsledkem dotazu př. B.7 je tak patrné, že datový typ není rozhodující pro shodu literálu.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?uri ?firstName
WHERE
{
  ?uri foaf:age "31".
  ?uri foaf:firstName ?firstName
}
```

Listing B.7: Dotaz na shodu literálu(číslo).

uri	firstName
<http://example.org/~miller/contact.rdf#petermiller>	„Peter“

Tabulka B.7: Výsledek dotazu 7

Literál v předchozím dotazu př. B.7 doplníme o datový typ. Na výsledek viz tab. B.8 však tato změna nemá žádný vliv. To znamená, že při dotazu nemusíme znát datový typ literálu a stačí znát jen samotný literál.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?uri ?firstName
WHERE
{
  ?uri foaf:age "31"^^xsd:integer
  ?uri foaf:firstName ?firstName
}
```

Listing B.8: Dotaz na shodu literálu s datovým typem.

uri	firstName
<http://example.org/~miller/contact.rdf#petermiller>	„Peter“

Tabulka B.8: Výsledek modifikovaného dotazu 7

B.1.8 Dotaz 8

Grafový vzor může být tvořen více než dvěma trojicemi, tuto vlastnost nyní ověříme. V dotazu př. B.9 se dotazujeme na URI, které má vlastnost *rdf:type* a hodnotu této vlastnosti *foaf:Person* a zároveň má další tři vlastnosti jejichž hodnoty chceme dotazem získat.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT $someuri ?name ?surname ?mbox
WHERE {
  ?someuri rdf:type foaf:Person
  ?someuri foaf:firstName ?name
  ?someuri foaf:lastName ?surname
  ?someuri foaf:mbox ?mbox
}
```

Listing B.9: Dotaz s 4 trojicemi v grafovém vzoru.

someuri	name	surname	mbox
<http://example.org/~miller/contact.rdf#petermiller>	„Peter“	„Miller“	<mailto:Miller.Peter@example.org>
<http://example.org/~austen#me>	„Austen“	„Lee“	<mailto:a.lee@example.org>

Tabulka B.9: Výsledek dotazu 8

B.1.9 Dotaz 9

Předchozí dotaz doplníme o klauzuli *OPTIONAL*, do této klauzule vložíme tvrzení, že nějaké URI má vlastnost *foaf:mbox*. Ve výsledku jsou tak zahrnuty i RDF grafy, které vlastnost *foaf:mbox* nemají.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT $someuri ?name ?surname ?mbox
WHERE {
  ?someuri rdf:type foaf:Person
  ?someuri foaf:firstName ?name
  ?someuri foaf:lastName ?surname
  OPTIONAL { ?someuri foaf:mbox ?mbox }
}
```

Listing B.10: Test klauzule OPTIONAL.

someuri	name	surname	mbox
<http://example.org/~miller/contact.rdf#petermiller>	„Peter“	„Miller“	<mailto:Miller.Peter@example.org>
<http://example.org/~austen#me>	„Austen“	„Lee“	<mailto:a.lee@example.org>
<http://www.example.com/lucy#contact>	„Lucy“	„O'Neal“	
<http://example.org/~jeremy/>	„Jeremy“	„Miller“	

Tabulka B.10: Výsledek dotazu 9.

B.1.10 Dotaz 10

Poslední testovací dotaz ověřuje funkčnost klauzulí *LIMIT* a *OFFSET*. Využijeme tak předchozí dotaz viz př. B.10 a ze znalosti výsledku předchozího dotazu ověříme, že nový dotaz vrací výsledky tab. B.11 dle očekávání.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT $someuri ?name ?surname ?mbox
WHERE {
  ?someuri rdf:type foaf:Person
  ?someuri foaf:firstName ?name
  ?someuri foaf:lastName ?surname
  OPTIONAL { ?someuri foaf:mbox ?mbox }
} LIMIT 2 OFFSET 1
```

Listing B.11: Test klauzulí LIMIT a OFFSET.

someuri	name	surname	mbox
<http://example.org/~austen#me>	„Austen“	„Lee“	<mailto:a.lee@example.org>
<http://www.example.com/lucy#contact>	„Lucy“	„O'Neal“	

Tabulka B.11: Výsledek dotazu 10.

B.2 Testy výkonnosti

Výkonnost RDF úložiště byla testována na 4 vzorcích dat(A,B,C,D) o počtu stovek(737), tisíců(4807), desetitisíců(41 847) a statisíců(176 291) RDF trojic. Pro každou sadu dat zde zmíním provedené dotazy a příslušné doby překladu a doby vykonání SQL dotazu. Zhodnocení výsledků je nabídnuto čtenáři v kapitole 7.2 o testování úložiště, tento dodatek jen doplňuje čísla a hodnocení o konkrétní podobu provedených dotazů.

B.2.1 Data A

- **Počet RDF trojic:** 737
- **Průměrný čas na vložení 1 RDF trojice** [s]**:** 0,001774

Dotaz I.a

```
PREFIX usgov: <http://www.rdfabout.com/rdf/usgov/geo/us/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?obj
WHERE {
  usgov:mt  dc:title ?obj
}
```

Listing B.12: Dotaz 1 nad sadou dat A.

- **Doba překladu** [s]**:** 0,001402
- **Doba vykonání SQL dotazu** [s]**:** 0,000249

Dotaz I.b

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?sub ?obj
WHERE {
  ?sub  dc:title ?obj
}
```

Listing B.13: Dotaz 2 nad sadou dat A.

- **Doba překladu** [s]**:** 0,001414
- **Doba vykonání SQL dotazu** [s]**:** 0,000254

Dotaz I.c

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?sub ?obj
WHERE {
  ?sub  dc:title ?obj
}
LIMIT 10
OFFSET 2
```

Listing B.14: Dotaz 3 nad sadou dat A.

- **Doba překladu** [s]**:** 0,001408
- **Doba vykonání SQL dotazu** [s]**:** 0,000297

Dotaz I.d

```
PREFIX tag: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub
WHERE {
  ?sub tag:landArea "1481346887193 m^2"
}
```

Listing B.15: Dotaz 4 nad sadou dat A.

- **Doba překladu** [s]**:** 0,001907
- **Doba vykonání SQL dotazu** [s]**:** 0,000235

Dotaz II.a

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX tag: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj1 ?obj2
WHERE {
  ?sub dc:title ?obj1
  ?sub tag:waterArea ?obj2
}
```

Listing B.16: Dotaz 5 nad sadou dat A.

- **Doba překladu** [s]**:** 0,002202
- **Doba vykonání SQL dotazu** [s]**:** 0,000272

Dotaz II.b

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX tag: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj1 ?obj2
WHERE {
  ?sub dc:title ?obj1
  OPTIONAL {
    ?sub tag:waterArea ?obj2
  }
}
```

Listing B.17: Dotaz 6 nad sadou dat A.

- **Doba překladu** [s]**:** 0,002339
- **Doba vykonání SQL dotazu** [s]**:** 0,000269

Dotaz II.c

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX tag: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj
WHERE {
  ?sub tag:waterArea "1847081495 m^2"
  ?sub dc:title ?obj
}
```

Listing B.18: Dotaz 7 nad sadou dat A.

- **Doba překladu [s]:** 0,002249
- **Doba vykonání SQL dotazu [s]:** 0,00022

Dotaz III.a

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX tag: <tag:govshare.info,2005:rdf/census/>
PREFIX govt: <tag:govshare.info,2005:rdf/usgovt/>
SELECT ?sub ?obj1 ?obj2 ?obj3 ?obj4
WHERE {
  ?sub dc:title ?obj1
  ?sub tag:landArea ?obj2
  OPTIONAL {
   ?sub tag:waterArea ?obj3
  }
  ?sub govt:uspsStateCode ?obj4
}
```

Listing B.19: Dotaz 8 nad sadou dat A.

- **Doba překladu [s]:** 0,003506
- **Doba vykonání SQL dotazu [s]:** 0,000303

Dotaz III.b

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX tag: <tag:govshare.info,2005:rdf/census/>
PREFIX govt: <tag:govshare.info,2005:rdf/usgovt/>
SELECT ?sub ?obj1 ?obj2 ?obj3 ?obj4
WHERE {
  ?sub dc:title ?obj1
  ?sub tag:landArea ?obj2
  ?sub tag:waterArea ?obj3
  ?sub govt:uspsStateCode ?obj4
}
```

Listing B.20: Dotaz 9 nad sadou dat A.

- **Doba překladu [s]:** 0,003404
- **Doba vykonání SQL dotazu [s]:** 0,000308

Dotaz IV. - popis zdroje

```
http://www.rdfabout.com/rdf/usgov/geo/us/wy
```

Listing B.21: Vyhledání trojic o určitém subjektu nad sadou dat A.

- **Doba překladu** [s]**:** 0
- **Doba vykonání SQL dotazu** [s]**:** 0,000214

B.2.2 Data B

- **Počet RDF trojic:** 4 807
- **Průměrný čas na vložení 1 RDF trojice** [s]**:** 0,002421

Dotaz I.a

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?obj
WHERE {
  <http://www.rdfabout.com/rdf/usgov/geo/us/wy/cd/110/1> dc:title ?obj
}
```

Listing B.22: Dotaz 1 nad sadou dat B.

- **Doba překladu** [s]**:** 0,0013
- **Doba vykonání SQL dotazu** [s]**:** 0,000212

Dotaz I.b

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?sub ?obj
WHERE {
  ?sub  dc:title ?obj
}
```

Listing B.23: Dotaz 2 nad sadou dat B.

- **Doba překladu** [s]**:** 0,000999
- **Doba vykonání SQL dotazu** [s]**:** 0,000342

Dotaz I.c

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?sub ?obj
WHERE {
  ?sub  dc:title ?obj
}
LIMIT 10
OFFSET 2
```

Listing B.24: Dotaz 3 nad sadou dat B.

- **Doba překladu [s]:** 0,00119
- **Doba vykonání SQL dotazu [s]:** 0,000242

Dotaz I.d

```
PREFIX census: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub
WHERE {
  ?sub  census:population "639087"
}
```

Listing B.25: Dotaz 4 nad sadou dat B.

- **Doba překladu [s]:** 0,0014
- **Doba vykonání SQL dotazu [s]:** 0,000208

Dotaz II.a

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX census: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj1 ?obj2
WHERE {
  ?sub  census:population ?obj1
  ?sub dc:title ?obj2
}
```

Listing B.26: Dotaz 5 nad sadou dat B.

- **Doba překladu [s]:** 0,001801
- **Doba vykonání SQL dotazu [s]:** 0,000401

Dotaz II.b

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX census: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj1 ?obj2
WHERE {
  ?sub  census:population ?obj1
  OPTIONAL {
    ?sub dc:title ?obj2
  }
}
```

Listing B.27: Dotaz 6 nad sadou dat B.

- **Doba překladu** [s]**:** 0,0019
- **Doba vykonání SQL dotazu** [s]**:** 0,0004

Dotaz II.c

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX census: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj
WHERE {
  ?sub dc:title  "Resident Commissioner District (at Large)"
  ?sub census:population  ?obj
}
```

Listing B.28: Dotaz 7 nad sadou dat B.

- **Doba překladu** [s]**:** 0,002001
- **Doba vykonání SQL dotazu** [s]**:** 0,000203

Dotaz III.a

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX tag: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj1 ?obj2 ?obj3 ?obj4
WHERE {
  ?sub dc:title ?obj1
  ?sub tag:landArea ?obj2
  OPTIONAL {
    ?sub tag:waterArea ?obj3
  }
  ?sub tag:population ?obj4
}
```

Listing B.29: Dotaz 8 nad sadou dat B.

- **Doba překladu** [s]**:** 0,003501
- **Doba vykonání SQL dotazu** [s]**:** 0,00046

Dotaz III.b

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX tag: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj1 ?obj2 ?obj3 ?obj4
WHERE {
  ?sub dc:title ?obj1
  ?sub tag:landArea ?obj2
  ?sub tag:waterArea ?obj3
  ?sub tag:population ?obj4
}
```

Listing B.30: Dotaz 9 nad sadou dat B.

- **Doba překladu** [s]**:** 0,003401
- **Doba vykonání SQL dotazu** [s]**:** 0,000422

Dotaz IV. - popis zdroje

```
http://www.rdfabout.com/rdf/usgov/geo/us/wa/cd/110/1
```

Listing B.31: Vyhledání trojic o určiitém subjektu nad sadou dat B.

- **Doba překladu** [s]**:** 0
- **Doba vykonání SQL dotazu** [s]**:** 0,000218

B.2.3 Data C

- **Počet RDF trojic:** 41 847
- **Průměrný čas na vložení 1 RDF trojice** [s]**:** 0,010186

Dotaz I.a

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?obj
WHERE {
  <http://www.rdfabout.com/rdf/usgov/geo/us/nc/counties/davidson_county> dc:title ?
obj
}
```

Listing B.32: Dotaz 1 nad sadou dat C.

- **Doba překladu** [s]**:** 0,001406
- **Doba vykonání SQL dotazu** [s]**:** 0,000241

Dotaz I.b

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?sub ?obj
WHERE {
  ?sub  dc:title ?obj
}
```

Listing B.33: Dotaz 2 nad sadou dat C.

- **Doba překladu [s]:** 0,000995
- **Doba vykonání SQL dotazu [s]:** 0,001116

Dotaz I.c

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?sub ?obj
WHERE {
  ?sub  dc:title ?obj
}
LIMIT 10
OFFSET 2
```

Listing B.34: Dotaz 3 nad sadou dat C.

- **Doba překladu [s]:** 0,001202
- **Doba vykonání SQL dotazu [s]:** 0,000231

Dotaz I.d

```
PREFIX census: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub
WHERE {
  ?sub  census:population "112249"
}
```

Listing B.35: Dotaz 4 nad sadou dat C.

- **Doba překladu [s]:** 0,001409
- **Doba vykonání SQL dotazu [s]:** 0,00021

Dotaz II.a

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX census: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj1 ?obj2
WHERE {
  ?sub  census:population ?obj1
  ?sub dc:title ?obj2
}
```

Listing B.36: Dotaz 5 nad sadou dat C.

- **Doba překladu [s]:** 0,001808
- **Doba vykonání SQL dotazu [s]:** 0,001529

Dotaz II.b

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX census: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj1 ?obj2
WHERE {
  ?sub  census:population ?obj1
  OPTIONAL {
    ?sub dc:title ?obj2
  }
}
```

Listing B.37: Dotaz 6 nad sadou dat C.

- **Doba překladu [s]:** 0,001909
- **Doba vykonání SQL dotazu [s]:** 0,001409

Dotaz II.c

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX census: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj
WHERE {
  ?sub dc:title  "Shasta County"
  ?sub census:population  ?obj
}
```

Listing B.38: Dotaz 7 nad sadou dat C.

- **Doba překladu [s]:** 0,001807
- **Doba vykonání SQL dotazu [s]:** 0,000237

Dotaz III.a

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX tag: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj1 ?obj2 ?obj3 ?obj4
WHERE {
  ?sub dc:title ?obj1
  ?sub tag:landArea ?obj2
  OPTIONAL {
    ?sub tag:waterArea ?obj3
  }
  ?sub tag:population ?obj4
}
```

Listing B.39: Dotaz 8 nad sadou dat C.

- **Doba překladu [s]:** 0,003504
- **Doba vykonání SQL dotazu [s]:** 0,002008

Dotaz III.b

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX tag: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj1 ?obj2 ?obj3 ?obj4
WHERE {
  ?sub dc:title ?obj1
  ?sub tag:landArea ?obj2
  ?sub tag:waterArea ?obj3
  ?sub tag:population ?obj4
}
```

Listing B.40: Dotaz 9 nad sadou dat C.

- **Doba překladu** [s]**:** 0,003409
- **Doba vykonání SQL dotazu** [s]**:** 0,00171

Dotaz IV. - popis zdroje

```
http://www.rdfabout.com/rdf/usgov/geo/us/wi/counties/douglas_county
```

Listing B.41: Vyhledání trojic o určiitém subjektu nad sadou dat C.

- **Doba překladu** [s]**:** 0
- **Doba vykonání SQL dotazu** [s]**:** 0,000245

B.2.4 Data D

- **Počet RDF trojic:** 176 291
- **Průměrný čas na vložení 1 RDF trojice** [s]**:** 0,0304

Dotaz I.a

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?obj
WHERE {
  <http://www.rdfabout.com/rdf/usgov/geo/us/al/counties/dekalb_county/crossville/
crossville> dc:title ?obj
}
```

Listing B.42: Dotaz 1 nad sadou dat D.

- **Doba překladu** [s]**:** 0,0017
- **Doba vykonání SQL dotazu** [s]**:** 0,00026

Dotaz I.b

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?sub ?obj
WHERE {
  ?sub  dc:title ?obj
}
```

Listing B.43: Dotaz 2 nad sadou dat D.

- **Doba překladu [s]:** 0,001005
- **Doba vykonání SQL dotazu [s]:** 0,39094

Dotaz I.c

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?sub ?obj
WHERE {
  ?sub  dc:title ?obj
}
LIMIT 10
OFFSET 2
```

Listing B.44: Dotaz 3 nad sadou dat D.

- **Doba překladu [s]:** 0,001305
- **Doba vykonání SQL dotazu [s]:** 0,00053

Dotaz I.d

```
prefix census: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub
WHERE {
  ?sub  census:households "986"
}
```

Listing B.45: Dotaz 4 nad sadou dat D.

- **Doba překladu [s]:** 0,00142
- **Doba vykonání SQL dotazu [s]:** 0,00024

Dotaz II.a

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX census: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj1 ?obj2
WHERE {
  ?sub  census:population ?obj1
  ?sub dc:title ?obj2
}
```

Listing B.46: Dotaz 5 nad sadou dat D.

- **Doba překladu [s]:** 0,00181
- **Doba vykonání SQL dotazu [s]:** 0,98082

Dotaz II.b

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX census: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj1 ?obj2
WHERE {
  ?sub  census:population ?obj1
  OPTIONAL {
    ?sub dc:title ?obj2
  }
}
```

Listing B.47: Dotaz 6 nad sadou dat D.

- **Doba překladu [s]:** 0,002095
- **Doba vykonání SQL dotazu [s]:** 1,06177

Dotaz II.c

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX census: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj
WHERE {
  ?sub dc:title  "Shasta County"
  ?sub census:population  ?obj
}
```

Listing B.48: Dotaz 7 nad sadou dat D.

- **Doba překladu [s]:** 0,002
- **Doba vykonání SQL dotazu [s]:** 0,000535

Dotaz III.a

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX tag: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj1 ?obj2 ?obj3 ?obj4
WHERE {
  ?sub dc:title ?obj1
  ?sub tag:landArea ?obj2
  OPTIONAL {
    ?sub tag:waterArea ?obj3
  }
  ?sub tag:population ?obj4
}
```

Listing B.49: Dotaz 8 nad sadou dat D.

- **Doba překladu [s]:** 0,003525
- **Doba vykonání SQL dotazu [s]:** 2,195895

Dotaz III.b

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX tag: <tag:govshare.info,2005:rdf/census/>
SELECT ?sub ?obj1 ?obj2 ?obj3 ?obj4
WHERE {
  ?sub dc:title ?obj1
  ?sub tag:landArea ?obj2
  ?sub tag:waterArea ?obj3
  ?sub tag:population ?obj4
}
```

Listing B.50: Dotaz 9 nad sadou dat D.

- **Doba překladu** [s]**:** 0,003655
- **Doba vykonání SQL dotazu** [s]**:** 2,03235

Dotaz IV. - popis zdroje

```
http://www.rdfabout.com/rdf/usgov/geo/us/al/counties/bibb_county/west_blocton/
west_blocton
```

Listing B.51: Vyhledání trojic o určiitém subjektu nad sadou dat C.

- **Doba překladu** [s]**:** 0
- **Doba vykonání SQL dotazu** [s]**:** 0,000575

Dodatek C

Obsah CD

Obsah CD je tvořen adresářovou struturou:

- `demo` - webová aplikace pro demonstraci knihovny RDFrepository
- `doc` - hypertextová dokumentace knihovny *RDFrepository* vygenerovaná programem phpDocumentor[1]
- `lib` - zdrojové kódy knihovny *RDFrepository* v jazyku PHP
- `sql` - sql skript pro vytvoření tabulek MySQL databáze a skript pro vložení testovacích dat viz př. B.1
- `bp.pdf` - technická zpráva v elektronické podobě

[1] phpDocumentor http://www.phpdoc.org/